ARTIFICIAL NEURAL NETWORK BASED BYZANTINE AGREEMENT PROTOCOL

(Version 2.0)

A Research-Prone

Master's Degree Thesis

by

Kok-Wah Lee @ Xpree Li

in

Computer Communications Security

URL: www.xpreeli.com

Copyright License of This Open-Source Book

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

_____

Lee Kok Wah @ Xpree Li Jinhua
Email: E96LKW@hotmail.com; xpree@yahoo.com; xpreee@gmail.com
URL: http://www.xpreeli.com; http://www.geocities.com/xpree/

25 Oct. 2002 (First Publication for version 1.0)
03 Jun. 2003 (First Revision)
02 Jun. 2008 (Second Revision)
10 Apr. 2010 (Second Publication for version 2.0)

ACKNOWLEDGEMENTS

It is my greatest pleasure to express my sincere gratitude to my nominal supervisor, Ewe Hong Tat (Mr.) (Dr.), from Faculty of Information Technology (FIT), Multimedia University (MMU) (Cyberjaya Campus), Malaysia. His understanding, sharp guidance, valuable encouragement and precious advice have made this project to be an enjoyable and successful tour of research. In addition, I wish to express my thanks to Professor Michael Fischer from Department of Computer Science, Yale University in New Haven, Connecticut, USA, for giving me his research paper.

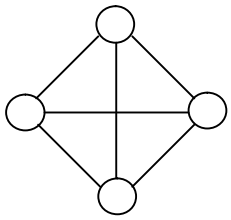
Concurrently, I am deeply in debt of gratitude to my parents for their love, trust, patience and understanding. Also, I wish to deliver my appreciation to grandma, uncles and aunts, as well as my cousins for their caring supports and mutual help. Last but no least, I would like to thank my friends and colleagues for their ideas, suggestions, jokes and supports in seeing the memorable outcome of this research journey.

Dedicated to

my beloved parents and family

who become the springs of my spirit.

H-(^_^)-H



ABSTRACT

The dependability of a distributed system is based on its reliability and availability, which are the fault-tolerant capabilities of a system in handling malfunctioning components and masking malicious faults. The spread of malevolent faults and software errors within the distributed systems and internet networks create a faulty situation where the faulty components will behave arbitrarily. This universal fault situation of a consensus problem is called the Byzantine Generals Problem (BGP) by Lamport, Shostak and Pease (1982). Since then, a number of Byzantine Agreement Protocols (BAPs) have been developed to solve the BGP.

In this thesis, an innovative and improved protocol to solve the BGP using the artificial neural networks is proposed. This new BAP, which is an ANN based BAP, has been tested its applicability through simulations over the connected network (FCN) models. This ANN based BAP has shown more advantages and great improvement over the traditional BAP. The advantages are less memory space demand, parallel processing at every node, and the dynamic learning ability of neural networks to Byzantine environment.

To accelerate the speed of achieving Byzantine Agreement (BA), the protocol is modified to reduce its rounds of message exchanges among the nodes to three rounds. The modified ANN based BAP has shown better performance over the traditional BAP when the size of the network $n \geq 10$. Furthermore, a mathematical analysis is carried out over the message exchange matrix formed during the message exchange stage. The analysis shows that the required mean epoch of neural network training decreases as the size of the network is increasing with fixed number of faulty nodes. In addition, the topic of faulty node detection is discussed. At last, ANN based BAP with 3-partition approach is also proposed to increase the performance of the ANN based BAP for about 10 times at the compensation of fewer faulty nodes.

Keywords: Artificial Neural Networks, Byzantine Generals Problem, Cryptography, Distributed System, Fault Tolerance, Network Security.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: OVERVIEW

1.1 MOTIVATIONS

The information era is started since the introduction of telephone, telegram and facsimile. This era booms when the computers are interconnected to form the computer networks. Via the Internet, the use of email, ftp and web applications have led to faster spread of globalisation. The advent of networking technologies has initiated the restructuring process of the traditional society model. Electronic commerce and electronic governance are to become parts of a new social model called *e-life* (Diffie & Hellman, 1976; Goldwasser, 1997).

In addition, many types of manual tasks nowadays are being replaced by the operations of computer systems. The computer systems have almost penetrated every possible aspects of human life. Almost all of these aspects are of utmost importance to avoid the occurrence of system failure. Avizienis (1976) had identified those failure cases to be avoided. These cases are as listed below:

i. The real time delays caused by manual repair after system failure are unacceptable. This is for systems with hard real time constraints, like process control applications, guidance systems, air traffic control systems, and fly-by-wire.

ii. It is impossible to manually repair the system. This is for systems that have to be unmanned, like systems used for space exploration.

iii. The costs of lost time and maintenance are excessively high. This is for systems like banking applications, life critical support systems, and defence systems.

Besides fault avoidance, the fault tolerance is introduced to enhance the computer system ability in handling system failure. Laprie (1985) brought forward the concept of *dependability*, which describes the reliability and availability of a computer system. We call this kind of computer systems as the dependable computer

systems. Among so many faults to be avoided and tolerated, the utmost arbitrary fault is Byzantine fault. It is a fault where the malicious components exhibit arbitrary behaviour to deliver totally different message to different components in a network.

Since then Byzantine fault has become a significant consensus problem in the distributed computing area. In 1976, Diffie and Hellman introduced the public key cryptosystem which helped ease some of the message delivery problems. In 1980, Pease, Shostak and Lamport proved that to solve this arbitrary fault consensus problem, the interactive consistency conditions (ICCs) have to be satisfied. Moreover, for a network with n processors and m faulty processors, n has to be at least $(3m+1)$ to ensure a possible BGP solution. Two years later, Fischer and Lynch showed that it needs a minimum of $(m+1)$ rounds of message exchanges to fulfil the ICCs by using the complex backward induction method. Applying an easy forward induction method, Aguilera and Toueg (1999) re-proved the $(m+1)$ requirement. Nevertheless, it is to note that the requirement of $(m+1)$ rounds of communication is derived on the assumption of single node message delivery.

In 1982, Lamport, Shostak and Pease presented a classical paper modelling the consensus problem with arbitrary fault, in analogy to an ancient Byzantine war, in a mathematical model called Byzantine Generals Problem (BGP). By applying the cryptography, they proposed a BGP solution called the traditional Byzantine Agreement Protocol (BAP) for both the oral message and written message. Nevertheless, since the traditional BAP uses the model of fully connected network (FCN) and Byzantine fault, it consumes a high degree of space and time resources in computing the Byzantine Agreement among the loyal components, though the system is not of high reliability requirement.

When the high reliability is not required, this kind of costly consumption is totally irrational. Consequently, higher fault classes are taken into mathematical modelling of the BGP instead of the universal fault class called Byzantine fault. After the traditional BAP (Lamport et al., 1982) that handles arbitrary processor fault in the FCN, there are a number of other BAPs being proposed. All of these BAPs can be identified from different assumptions on network topology, processor fault and link fault. Table 1.1 lists nine BAPs categorised according to the assumptions.

The first BAP (Lamport, Shostak & Pease, 1982) and the second BAP (Dolev, 1982) treat all faults as the arbitrary faults. In spite of the existence of some faults as dormant faults, such as fail-stop and crash faults, both BAPs ignore the fact that the dormant faults are better than the arbitrary faults. When a dormant fault exhibits its faulty behaviour, it can be detected and ignored by all loyal processors. The first BAP bases on a FCN model, whereas the second BAP bases on a non-FCN model. The weakness of these two BAPs is that both fail to handle the maximum number of faults if the dormant faults exist. For the third BAP (Christian, Aghili & Strong, 1985), it is a BAP handling only dormant faults based on the FCN model. Its disadvantage is that though it can cope with more number of faults, the existence of a single arbitrary fault can fail the protocol.

In view of the lack of BAP to handle both arbitrary faults and dormant faults, some BAPs are created to cope with both of them. BAP handling the mixed fault is a hybrid fault model allowing the maximum number of faulty processors to be tolerated. The forth (Thambidurai & Park, 1985), fifth (Meyer & Pradhan, 1991) and sixth (Lincoln & Rushby, 1993) BAPs are all hybrid fault models. The forth and sixth BAPs are based on FCN model handling hybrid faults. Both BAPs assume that the number of processors suffering from arbitrary faults must be known prior to the implementation of the protocols.

Nevertheless, this condition violates the BGP assumption that a loyal processor is ignorant to the faulty status of other processors. Other than the violation, it is impractical to run diagnostics for detecting all the Byzantine faults in a network (Shin & Ramanathan, 1987). Meyer and Pradhan (1991) discovered another weakness of hybrid fault model. The hybrid fault model is unable to reach Byzantine Agreement among processors when the number of arbitrary faults is underestimated or overestimated. Meyer and Pradhan (1991) proposed two BAPs (Mixed BAP & Mixed-Sum BAP) to help improve the hybrid fault models. Both Mixed BAP and Mixed-Sum BAP are designed for non-FCN model. The Mixed BAP needs to know the number of arbitrary faults before initialization. On the contrary, the Mixed-Sum BAP is designed to overcome this limitation.

Table 1.1 Various BAPs according to different assumptions

No.	Assumptions	Network Topology			Processor Fault			Link Fault	
	BAPs	FCN	Non-FCN	Broadcast	Arbitrary	Dormant	Hybrid	Arbitrary	Hybrid
1	Lamport, Shostak & Pease (1982)	X			X				
2	Dolev (1982)		X		X				
3	Christian, Aghili & Strong (1985)	X				X			
4	Thambidurai & Park (1988)	X					X		
5	Meyer & Pradhan (1991)		X				X		
6	Lincoln & Rushby (1993)	X					X		
7	Babaoglu & Drummond (1985)			X	X			X	
8	Yan & Chin (1988)	X						X	
9	Yan, Chin & Wang (1992)	X			X			X	
10	Siu, Chin & Yang (1998)		X				X		X
11	Wang & Kao (2001)	X			X				

To simulate the fault model at a higher degree, BAP is then designed to consider the link faults. Babaoglu and Drummond (1985) suggested the seventh BAP where a broadcast network is used for handling both arbitrary processor faults and arbitrary link faults. Yan and Chin (1988) proposed a BAP using the FCN to tolerate only arbitrary link faults. In 1992, Yan, Chin and Wang extended the previous BAP to tolerate both arbitrary processor faults and arbitrary link faults. Then, Siu, Chin

and Yang (1998) recommended a BAP tolerating hybrid processor faults and hybrid link faults. All of the BAPs discussed so far are of serial processing at each processor. It means every processor has to wait for its turn before the other processor has finished its task of message delivery. This kind of information processing is not only time consuming, but also requires a large amount of space for data storage.

In order to solve the weakness of serial processing, Wang and Kao (2001) suggested a new approach to Byzantine Agreement by using the artificial neural network (ANN). It lays the base on FCN to tolerate arbitrary processor faults. Due to the natural property of ANN, parallel processing at each node is then successfully achieved. The time and space requirements are greatly reduced. Furthermore, the adaptability of ANN allows the learning capability of this BAP to flexible Byzantine environment.

In this thesis, further improvements are implemented on the BAP proposed by Wang and Kao (2001). The BAP is called the artificial neural network based Byzantine Agreement Protocol (ANN based BAP) and applies traditional back propagation neural network (BPNN) approach into the artificial neural network. In Lee and Ewe (2001), detailed analysis and discussion of ANN based BAP were presented. In addition, more experiments are run on the model of FCN-*n*. The experiments are carried out on neural networks trained by both the traditional BPNN and modified BPNN techniques.

Wang and Kao's BAP (2001) uses (m+1) rounds of message exchanges to form the message exchange matrix for neural network training. This is because the message interchanged among each other node is of single node value. In this research, the BAP efficiency is increased to 3 rounds by cutting the number of rounds of communication (Lee & Ewe, 2002a). This is possible by adopting the technique used both in Yan, Wang and Chin (1999), and Wang and Yan (2000). This technique delivers single node value at the first round and second round, but sends out a string values of other nodes in the third round.

From the experimental results in this research project (Chapter 4), it is found that the modified BPNN gives better training results. It needs less number of epochs as compared to traditional BPNN. Consequently this allows faster achievement of

Byzantine Agreement. Moreover, the results show that the larger the network the less the epoch is for fixed number of faulty nodes (Lee & Ewe, 2002b). This phenomenon is then explained by using an analysis on message exchange matrix. The analysis shows that the bit composition property is the cause of this phenomenon. Besides the mentioned phenomenon, the bit composition of the message exchange matrix allows the detection of the faulty nodes.

Lastly in this research project, a new approach which is the ANN based BAP with 3 partitions is introduced to further increase the performance of BAP with no partition. In this proposed BAP, a network of nodes is partitioned into three groups of nodes. Every group has a trusted party for the nodes to rely on. From the analysis, the speed of achieving the Byzantine Agreement increases for about ten times as compared to the ANN based BAP with no partition. Nevertheless, this improvement is based on the compensation of maximum possible number of faulty nodes to be tolerated in a malicious network. In addition, other types of partition are also discussed to show that ANN based BAP with 3 partitions is in fact an optimized solution.

1.2 SCOPE OF THE THESIS

In the beginning of this research project, a review has been carried out over the fault-tolerant distributed systems. The dependability concept, fault classification and fault tolerance in the distributed systems are discussed in Chapter 2. Then the solution of the universal fault class namely Byzantine fault is documented under the subtopics of Byzantine Generals Problem (BGP) and traditional Byzantine Agreement Protocol (BAP). In this chapter too, two examples of traditional BAP are presented together with the analysis of its complexity.

In Chapter 3 of this thesis, the new approach to BAP using artificial neural network is presented. The background, applications and advantages of ANN are presented in the same chapter. In Chapter 4, the ANN based BAP is discussed in details on the subtopics of implementation, complexity and advantages.

In Chapter 5, an analysis of the message exchange matrix (MEM) has been done to explain the phenomenon of fewer epochs for larger network with fixed

number of faulty nodes. In the following Chapter 6, the MEM is further discussed to show its ability of faulty node detection.

In Chapter 7, a new type of ANN based BAP is proposed. The previous few chapters discuss ANN based BAP with no partition, but this chapter investigates the possibility of introducing partitions into a malicious network. After the optimization, the ANN based BAP with 3 partitions is proposed together with its advantages and compensations. It is then compared to ANN based BAP with no partition and ANN based BAP with k partitions. In Chapter 8, the summary of this research project and recommendations of future research directions are included.

CHAPTER 2: BYZANTINE GENERALS PROBLEM IN A DISTRIBUTED FAULT-TOLERANT SYSTEM

2.1 INTRODUCTION

In December 1947, J. Bardeen, W. Brattain and W. Shockley from Bell Lab invented the transistor. Nevertheless, no one has predicted the emergence of the computer and the Internet in less than half a century. The series of inventions has brought the human civilisation to a new era called information age. The omnipresence of the computer systems in the society nowadays generates the demand for highly dependable distributed systems. Among the dependable systems, they are divided into the non-redundant systems and the redundant systems.

Fault-tolerant distributed system, which is a subset of the redundant systems, will be discussed in this chapter in addition to the dependability measures of a distributed computing system. Then the universal fault class, i.e. *Byzantine fault*, will be discussed under the topic of *Byzantine Generals Problem* (BGP). The first traditional method (Lamport, Shostak & Pease, 1982) to solve the BGP is presented under the topic of *Traditional Byzantine Agreement Protocol* in Section 2.6.

2.2 DEPENDABILITY

The increasing computing power and memory space of the computers have allowed the possibility for computers to replace a majority of the manual tasks. These tasks include the manufacturing processes, banking, flight control systems, patient monitoring systems, ticket booking systems, traffic control systems and the centralized control systems of high rise buildings. Computers have penetrated almost all the possible aspects of a human living in a modern society today. This situation makes a modern society to be more and more dependable on the proper functioning of distributed computer systems. The tremendously increasing utilization rate of computer system has triggered off the essential and critical requirements for highly dependable distributed computer systems.

The concept of dependability is initially introduced by Laprie (1985) in an attempt to create a consistent terminology environment in the area of reliable computing. *Dependability* is defined as the property of a computer system such that reliance can be justifiably placed on the service it delivers (Laprie, 1995). The *service* delivered by a system is its behaviour from the perception of its users. A *user* is another physical or human system interacting with the former. In Laprie (1985, 1992, 1995), the dependability concept had been distinguished from the aspects of *attributes*, *impairments*, *measures* and *means*. In 1994, Jalote explained the dependability and its aspects in details.

2.2.1 Dependability Attributes

To evaluate the degree of dependability of a distributed system, different attributes are used according to the application of the system. In Laprie (1995), six most significant attributes were chosen. They are availability, reliability, safety, confidentiality, integrity and maintainability.

Availability means readiness for usage. *Reliability* means the continuity of service delivery. *Safety* means the non-occurrence of the catastrophic consequences on the environment. *Confidentiality* is the non-occurrence of the unauthorised disclosure of information. *Integrity* is the non-occurrence of improper alterations of information. Lastly, *maintainability* is the ability to undergo the repairs and evolutions. The fourth and the fifth dependability attributes are both the properties of cryptology.

2.2.2 Dependability Impairments

In addition to attributes, dependability of a system is illustrated by its impairments. *Impairments* consist of faults, errors and failures. Increasing the dependability of a distributed system helps minimise the probability of occurrence of a *system failure*. A system failure happens when the delivered service first deviates from the required system specifications (Anderson & Lee, 1981). Failure occurs when the system responses to an input in a way different from the user expectation.

An *error* is a part of the system state that is liable to lead to subsequent failure. The cause of an error is a fault (Jalote, 1994). If there is an error in the system state, then there exists a sequence of actions which can be executed by the system to avoid a system failure. Hence, the failure is caused by the error directly and the fault indirectly in the hardware components or the software processes of the distributed system.

Fault is defined as the defects having potential of generating errors. A faulty system is a system that is with defects. A fault can be caused by physical defects of electrical components, execution of incorrectly designed programs, or operator mistakes. During the observation period, a fault may or may not generate an error. For example, consider a faulty memory cell which always returns a value of bit "1" regardless of the value stored. The cell faulty state may not manifest itself if all the while only bit "1" is stored in it. In this case, though the cell is faulty, the retrieved value is correct and no error exists in the system. The faulty cell will only generate an error when bit "0" is stored but the retrieval is a bit "1" instead of a bit "0".

2.2.3 Dependability Measures

Dependability *measures* are used to quantify the system dependability. The adequacy of a measure to express dependability depends on the specific application that is being used. For instance, the reliability is quantified by using the *mean time to failure (MTTF)* or *expected life* in Jalote (1994). The MTTF is expressed as a function *R(t)*, which is the probability that a system still survives at time *t*. From time *0* till time *t*, if the system has not failed, then the MTTF is given by Equation (2.1).

$$MTTF = \int_0^t R(t)dt \tag{2.1}$$

Other than the reliability, another driving force for fault tolerance is the availability. Jalote (1994) defined the *instantaneous availability A(t)* of a component to be a probability that the component is functioning correctly at time *t*. In the absence of repair or replacement, availability is simply the same with reliability. With replacement, the component life can be considered as a sequence of

independent random variables each representing the life of the component till the next failure and repair. Among the variables, we have a random variable T_i to represent the duration for the functioning period for *i*-th component, and another random variable D_i to represent the downtime for the repair or replacement for *i*-th component. Normally, a system availability is analyzed in term of *limiting availability*, which is the limit of *A(t)* as *t* approaches infinity. For reliability, its limiting value is zero as *t* approaches infinity. However, the limiting availability is typically more than zero. Let *MTTF* be the mean time to failure of a component (where each T_i has *MTTF* as the mean), and *MTTR* be the mean time to repair for the component (where each D_i has *MTTR* as the mean). The *(limiting) availability*, α, will be given by Equation (2.2) (Trivedi, 1982). In other words, availability means the ratio of time the system is available for useful operation.

$$\alpha = \lim_{t \to \infty} A(t) = \frac{MTTF}{MTTF + MTTR} \tag{2.2}$$

2.2.4 Dependability Means

The *means* of dependability are the techniques used for improving the dependability of a distributed system. Specifically, means are to ameliorate several important dependability attributes. The most important attributes for reliable computing so far in a distributed system are reliability and availability. Among so many techniques, there exist four groups of techniques for a system to improve its reliability and availability. They are *fault avoidance*, *fault detection*, *masking redundancy*, and *dynamic redundancy*. The application of the techniques by a dependable system will determine whether it is a non-redundant system or a redundant system. For system with only the fault avoidance technique, it is a non-redundant system. Meanwhile the other three techniques make a dependable system to be a redundant system. For the techniques of masking redundancy and dynamic redundancy, the redundant system is specifically known as fault-tolerant system. Figure 2.1 gives an overview of a dependable system (Postma, 1998).

Fault avoidance or *fault prevention* is a direct approach to ameliorate the reliability and the availability of a system. This approach tries to prevent fault from

occurring or being introduced into the system. For a highly reliable and available system, fault avoidance tries to remove as many faults as possible before the system is in use. The system software and the behaviour of all the hardware components are extensively tested. Since there is no redundant software program nor hardware component within the system, the system using only the fault avoidance technique is a non-redundant system. As in the study by Krol (1991), this technique improves the system reliability by a factor of 10 with relative cheap cost.

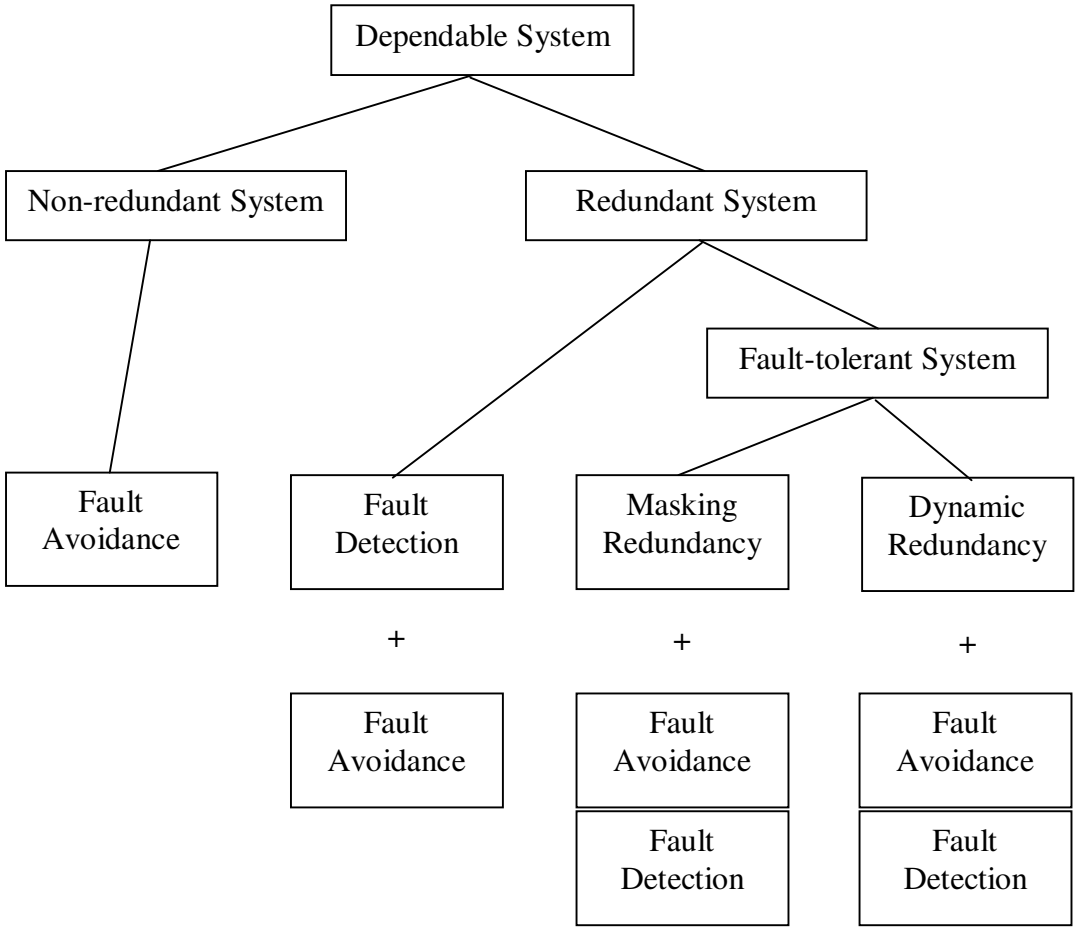


Figure 2.1 Overview of dependable system

In system using the *fault detection*, redundancy exists to detect the faults. Upon the detection of the faults, the system reports the fault occurrence, and then the system is shut down, diagnosed, and manually reconfigured for a restart later. In the

study by Siewiorek and Swarz (1998), system using the fault detection technique was usually not considered as fault-tolerant system due to its incapability to tolerate any type of fault. The hardware types of fault detection techniques are duplication, error detecting codes, checksums, self-checking and fail-safe logic, watchdog timers and bus timeouts, consistency and capability checks, and processor monitoring. Meanwhile the software type of fault detection technique is the program monitoring.

To improve the reliability and availability of a non-redundant dependable system, the application of multiple fault tolerance techniques is required. However the application of both the fault detection and fault avoidance techniques is not sufficient. This is because a system based merely on the fault detection and/or fault avoidance can only afford a guaranteed service until the occurrence of a fault. On the other hand, a system based on masking redundancy and/or dynamic redundancy may continue its service upon the occurrence of one or more faults in the system. Therefore, system applying the masking redundancy and/or dynamic redundancy techniques is called as a *fault-tolerant system*.

Masking redundancy is also known as static redundancy. It is applied to mask component failure in a system. The system continues its service if the number of failed components stays below a predefined maximum. For *dynamic redundancy*, the system is dynamically reconfigured upon the component failures. This reconfiguration allows the replacement of failed components by correctly functioning spare components. A further discussion on fault tolerance is in Section 2.4.

2.3 FAULT CLASSIFICATION

The faults in a system can be classified in two ways. The first one is from the viewpoint angle by Siewiorek and Swarz (1998), and the second classification method is from the fault behaviour of the components (Cristian, 1991; Barborak, 1993; Alstein, 1996).

Figure 2.2 shows the fault classification according to different viewpoints. From the viewpoint of *pattern of occurrence*, there are three classes of faults: permanent, intermittent and transient faults. A *transient fault* is a fault of short duration caused by temporary environmental conditions. An *intermittent fault* is a

transient fault occurring repeatedly or occasionally present due to unstable hardware or varying hardware or software states. A clear difference between transient fault and intermittent fault is that the intermittent fault is detectable and repairable by replacement or redesign, whereas it is hard to detect a transient fault because the hardware is physically undamaged. *Permanent fault* is a continuous and stable fault due to an irreversible physical change. In other words, once the component fails, it never works correctly again.

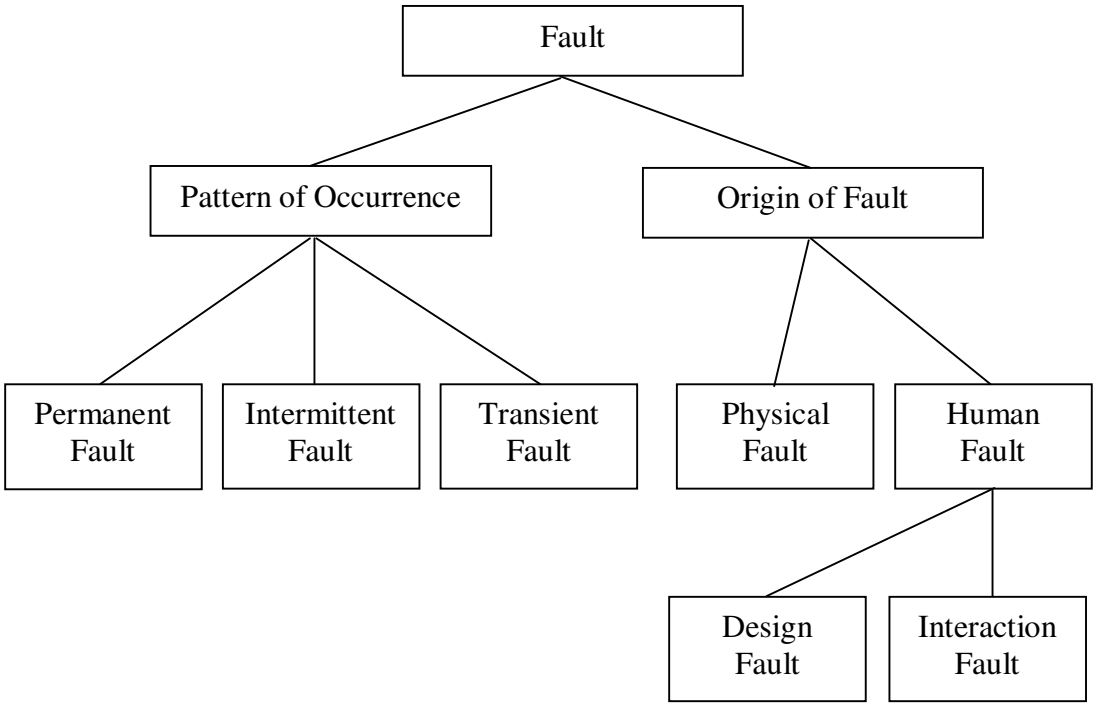


Figure 2.2 Fault classification according to different viewpoints

From the viewpoint of *origin of fault*, there are physical fault and human fault. A *physical fault* is a fault stemming from physical phenomena internal to the system (threshold changes, shorts or opens) or external changes (environmental, electromagnetic or vibrating effects). *Human fault* is either design fault or interaction fault caused by human. *Design fault* is committed during the system design, modification or establishment of operating procedures. *Interaction fault* or *operational fault* occurs during the system lifetime due to violations of operation or

maintenance procedures. In general, interaction fault is easier to be tolerated than design fault (Siewiorek & Swarz, 1998; Jalote, 1994).

On the other hand, there is another way to classify the fault in an ordered arrangement of fault classes. Every fault class is determined by its own fault behaviour within the class. *Fault behaviour* is the way a component may exhibit once it is faulty. This classification method describes the fault behaviour of components mathematically to separate the faults into ordered fault classes. There is a scope of allowed faults in each fault class. All the fault classes are arranged from the strongest to the weakest. As Barborak, Malek and Dahbura (1993) demonstrated, the fault classes could be classified into seven groups starting from the strongest: fail-stop faults, crash faults, omission faults, timing faults, incorrect computation faults, authenticated Byzantine faults, and Byzantine faults. It is to note here that every stronger fault class is the subset of a weaker class. Lamport, Shostak and Pease (1982) found that the stronger an assumed fault model is, the easier to take this fault behaviour into account for achieving the solution. Figure 2.3 shows the ordered fault classification.

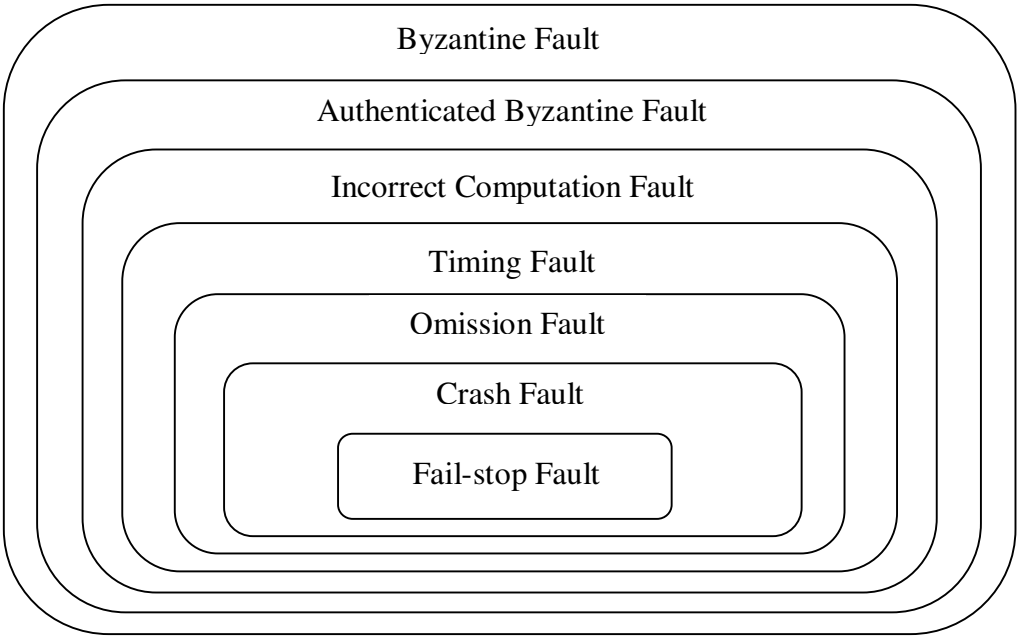


Figure 2.3 An ordered fault classification

The *fail-stop fault* belongs to a fault class containing any fault that occurs when a processor ceases operation and alerts other processors of this fault (Schlichting & Schneider, 1983). The class of *crash fault* consists of any fault that occurs when a processor loses its internal state or halts. The class of *omission fault* is a fault class that occurs when a processor omits to respond to an input (Cristian, 1991). The class of *timing fault* comprises any fault that occurs when a processor fails to complete its task within the specified time frame (completing the task after its specified time frame or never) (Cristian, Aghili, Strong & Dolev, 1995).

The class of *incorrect computation fault* is a class of any faults that occurs if a processor fails to produce the correct result in response to the correct inputs (Laranjeiri, Malek & Jenevein, 1991). According to Barborak, Malek, Dahbura (1993) and Laranjeiri, Malek, Jenevein (1991), the incorrect computation fault class was a superset of the fail-stop, crash, omission and timing fault classes and a subset of the Byzantine fault class. The *authenticated Byzantine fault* class consists of any arbitrary or malicious fault except that the faulty processors are incapable of imperceptibly altering a message signed by a correct processor. A faulty processor may collude with another faulty processor (Lamport, Shostak & Pease, 1982). The *Byzantine fault* class has every arbitrary or malicious fault that is possible in a system (Lamport, Shostak & Pease, 1982). This fault class is a universal fault set.

2.4 FAULT TOLERANCE

As in Section 2.2.4, a fault-tolerant system is a system employing the techniques of masking redundancy and dynamic redundancy. Masking redundancy is also known as *static redundancy*. The hardware techniques based on masking redundancy are N-modular redundancy, error correcting codes and masking logic, whereas the software technique is N-version programming. For hardware techniques based on dynamic redundancy, they are re-configurable duplication, re-configurable N-modular redundancy, backup sparing and graceful degradation, whereas the software techniques are backward and forward error recovery.

A system is *fault tolerant* if it can mask the presence of faults in the system and prevent the system from system failure in the presence of component failure. In

other words, it means a system is considered as fault tolerant if the system behaviour is consistent with its specifications in spite of the component failure (Jalote, 1994). To achieve fault tolerance, a distributed system architecture incorporates redundant processing components (Cristian, 1991). *Redundancy* is defined as those parts of the system not needed for the correct functioning of the system when there is no fault tolerance to be supported. It means the system can work without the existence of redundancy whenever no failure occurs. There are three types of system redundancy: hardware, software and time (Avizienis, 1976).

Hardware redundancy consists of the hardware components that are added to tolerate the system from faults. The examples of hardware redundancy are RAID disks and backup name servers. *Software redundancy* comprises all program and instructions employed for supporting the fault tolerance. Data replication and coding are all examples of software redundancy. The examples of software redundancy using the coding technique are Hamming code, parity memory, error correcting code (ECC) on data block and memory. *Time redundancy* is a fault-tolerant technique employing extra time for performing the fault-tolerant tasks. The examples of time redundancy are executing the instructions for a few times and performing an operation several times. A fault-tolerant system without hardware redundancy cannot tolerate any permanent fault. Meanwhile the time redundancy is useful for both transient and intermittent faults, but it is totally useless to permanent faults.

When a fault occurs in a redundant system, there are a number of steps to be followed to treat the fault. Siewiorek and Swarz (1998) had suggested a fault treatment recipe comprising seven steps: fault detection, fault diagnosis, re-configuration, recovery, restart, repair and reintegration. *Fault detection* is to detect unexpected events happening in the system. *Fault diagnosis* is to obtain the information about the location and the nature of the detected fault. *Re-configuration* is involved when there is a permanent component failure. The failed component will be replaced with a backup spare by re-configuring the system. *Recovery* eliminates the effects of faults, either by means of techniques of fault masking or retry. *Restart* is to initialize the system again after the recovery done. In the *repair* step, a failed component is replaced either online or offline. In *reintegration*, the repaired component is integrated to the system again.

2.5 BYZANTINE GENERALS PROBLEM (BGP)

In Section 2.3, the concept of Byzantine fault is introduced as a universal fault set. Jalote (1994) defined the *Byzantine fault* as an arbitrary fault, which caused the component to behave in a totally arbitrary manner during failure. In this section, the Byzantine fault will be further discussed. To tolerate the Byzantine fault, Lamport, Shostak and Pease (1982) firstly described it as a mathematical model called Byzantine Generals Problem (BGP). They had linked the BGP with the problem faced by the reliable computer system in handling malfunctioning components that delivered conflicting information to different parts of the system. Hence, BGP is actually a type of consensus problems.

Byzantine Generals Problem is named after a martial situation of an ancient war in the Middle Ages. It is a situation where a group of generals of the Byzantine army besieges around an enemy city. Those Byzantine generals have to achieve a consensus on a common battle plan via the communication of the messengers only. Nevertheless, one or more of the Byzantine generals may be traitors who will try to confuse the other loyal Byzantine generals with an aim of letting the Empire of Byzantium fails the battle. Here comes the problem called BGP where the Byzantine generals have to find an algorithm to ensure that all the loyal Byzantine generals will achieve the same agreement after the communication of the messengers among each other, in spite of the confusion effort done by the faulty generals.

In the context of a distributed computer network, there is an analogy between malicious attacks, software errors, and the mentioned situation of the Byzantine generals. The spread of the malicious faults and software errors within the internet network and distributed system create a faulty situation where the faulty components will behave arbitrarily. Under this malicious environment, both the loyal and faulty components exist, and the loyal components are always trying their best to mask the malicious faults from the other faulty components. In the scope of a distributed system, a component is in fact representing a processor. This malicious situation is called as *Byzantine environment* where the faulty processors send out arbitrary information known as Byzantine fault to other processors within the computer system or distributed network.

A distributed system comprises a set of connected processors via a communication network. The processors can be viewed as nodes and the links between processors as paths of a graph. Hence a network problem is simplified to become a graph problem. Among the nodes, both the loyal and faulty nodes exist.

For fault tolerance within a system or network, the method of redundancy is always adopted. It means multiple nodes in a network implement the same task and the results are compared with each other to arrive at a majority vote. With this simple method, different fault classes can be detected, masked and tolerated. However, one kind of fault cannot be tolerated so easily. There is no way to detect a faulty node if the node is still answering on the network but the answers are wrong. This rough and very mathematical consensus problem is then developed to become a more concrete problem called *Byzantine Generals Problem*. This problem treats the nodes in the network as commanders of small armies. The armies consist of lieutenants who get an order from a commanding general. All the faulty nodes are considered as traitors.

To solve the BGP, all the loyal nodes have to achieve the same agreement among them in spite of the disturbance from the faulty nodes. The agreement was named as *Byzantine Agreement* (BA) by Feldman (1988). In a majority of the situations, the values to be mutually agreed may have arbitrary length. However, it was known that the problem of achieving Byzantine Agreement on arbitrary values could be easily reduced to a problem of reaching agreement on binary value as in Feldman's doctoral dissertation (1988). The solution of BGP or the procedure to arrive at the Byzantine Agreement is called Byzantine Agreement Protocol (BAP). The relationship of BA and BAP is given as follows:

Let P be a protocol of solving the BGP and B_i be the private input bit of each loyal node i. When each loyal node i terminates correctly after the message exchange phase, let d_i be the final output bit of the loyal node. Then P is a Byzantine Agreement Protocol (BAP) if P fulfils both of the conditions of (A.1) and (A.2).

(A.1) For any node i and node j that terminate correctly, $d_i = d_j$.

(A.2) If $B_i = B_j$ for all the loyal nodes, then for each node i that terminates correctly, $d_i = B_i$.

Feldman (1983) called the consensus of the BGP with one-bit agreement as BA, and the consensus of the BGP with multi-bit agreement as *general BA*. For general BAP or multi-bit BAP, it is defined as below:

Let P be a protocol of solving the BGP and X_i be the private input string of each loyal node i. When each loyal node i terminates correctly after the message exchange phase, let y_i be the final output string of the loyal node. Then P is a general Byzantine Agreement Protocol (general BAP or multi-bit BAP) if P fulfils both of the conditions of (B.1) and (B.2).

(B.1) For any node i and node j that terminate correctly, $y_i = y_j$.

(B.2) If $X_i = X_j$ for all the loyal nodes, then for each node i that terminates correctly, $y_i = X_i$.

In the classic paper "The Byzantine Generals Problem" by Lamport, Shostak and Pease (1982), the unanimity problem of BGP was further divided into two categories depending on the form of the message being exchanged among the nodes in a network. The first type is called BGP with oral message (or unsigned message) and the second type is called BGP with written message (or signed message). Nevertheless, in order to achieve the BA, both types of the BGP have to satisfy the conditions of (C.1) and (C.2) described below.

In a *Byzantine Generals Problem* with n generals, a commanding general must send an order to his (n-1) lieutenant generals so that:

(C.1) All loyal lieutenants obey the same order.

(C.2) If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Conditions (C.1) and (C.2) are named as *interactive consistency conditions* (ICCs) (Pease, Shostak & Lamport, 1980). It is easy to notice that if the commander

is loyal, then (C.1) follows from (C.2). Anyway, the commander needs not to be loyal. In analogy, the BAP for BGP with both loyal nodes and faulty nodes in a network must satisfy the ICCs of (D.1) and (D.2) as below:

In a *Byzantine Generals Problem* with *n* nodes within a network, a source node must send a message to its (*n*-1) lieutenant nodes so that:

(D.1) All loyal lieutenant nodes obey the same order.

(D.2) If the source node is loyal, then every loyal lieutenant node obeys the order it sends; or DEFAULT value is used if it receives no value.

(C.1) or (D.1) is normally known as *agreement* of the ICCs, and (C.2) or (D.2) is normally known as the *validity* of the ICCs. To achieve the ICCs, every lieutenant node delivers its message to every other lieutenant until everyone obtains a message from everyone else. Then every node decides upon the common agreement, which is normally the majority decision. Consequently, all the loyal nodes have achieved the Byzantine Agreement in unison.

2.5.1 System Model

Before the BGP of oral message and written message is discussed, it is good to understand the possible network model of a distributed system. This will help in the simplification of a complex system model. A *system* is defined as an identifiable mechanism, which maintains a pattern of behaviour at an interface between the system and its environment (Anderson & Lee, 1981).

A distributed system consists of many computers that are geographically located but are connected by a communication network. The computers (or processors) are represented as nodes, and the links between any two computers are represented as paths. All the nodes are autonomous and communicate with each other by exchanging messages over the communication network. Via this representation, a computer network or distributed system can be modelled as a graph. It is to take note

here that the key properties of distributed system are the geographical separation and autonomous nature of various nodes.

This network model is important since it helps describe the relationship and interaction among any party. However most of the network protocols are 2-party protocols, which offer neighbour to neighbour or end system to end system communication. This kind of traditional network layer protocol can only help ease the simple failure such as inoperative nodes or links. In contrast, the existence of arbitrary behaviour or Byzantine failure, such like corrupting, forging, or delaying routing protocol messages, in a distributed system requires a multi-party protocol. This multi-party protocol requires the cooperation of all the loyal nodes to mask or to tolerate any Byzantine fault. BAP is the protocol that helps solve the BGP in the network layer (Perlman, 1988).

For system model, there are physical model and logical model. Physical model is from the point of view of the physical components of the system. Logical model is from the point of view of processing or computation. These models have their nodes to communicate with each other via a communication network. The communication network comprises a number of communication links, which are normally point-to-point type of links. Other types are bus network and ring network.

There exist many ways of connections between the nodes in a communication network. The architecture of the network connections is called as *network topology*. For point-to-point network, various types of network topologies are possible. For example, the common network topologies are the star, the tree and the fully connected network (FCN). FCN is a type of communication network model where each node is connected to all the other nodes via a dedicated link. Figure 2.4 shows the different types of communication network topologies.

In a practical world, it is essential to take into consideration the assumption for the time bounds on the performance of the system. A system is defined as *synchronous* if whenever the system is working correctly, it always performs its intended function within a finite and known time bound. If a system exhibits irregular or unpredictable response time, it is said to be an *asynchronous* system (Mishra & Schlicthing, 1992). A synchronous communication channel or link

between two nodes is one in which the maximum message delay is known and bounded. A synchronous processor or node is the one in which the time to execute a sequence of instructions is finite and bounded. A synchronous system allows the failure detection of lack of response within some defined time bound. In this thesis, all the links and nodes in the distributed system are assumed to be synchronous.

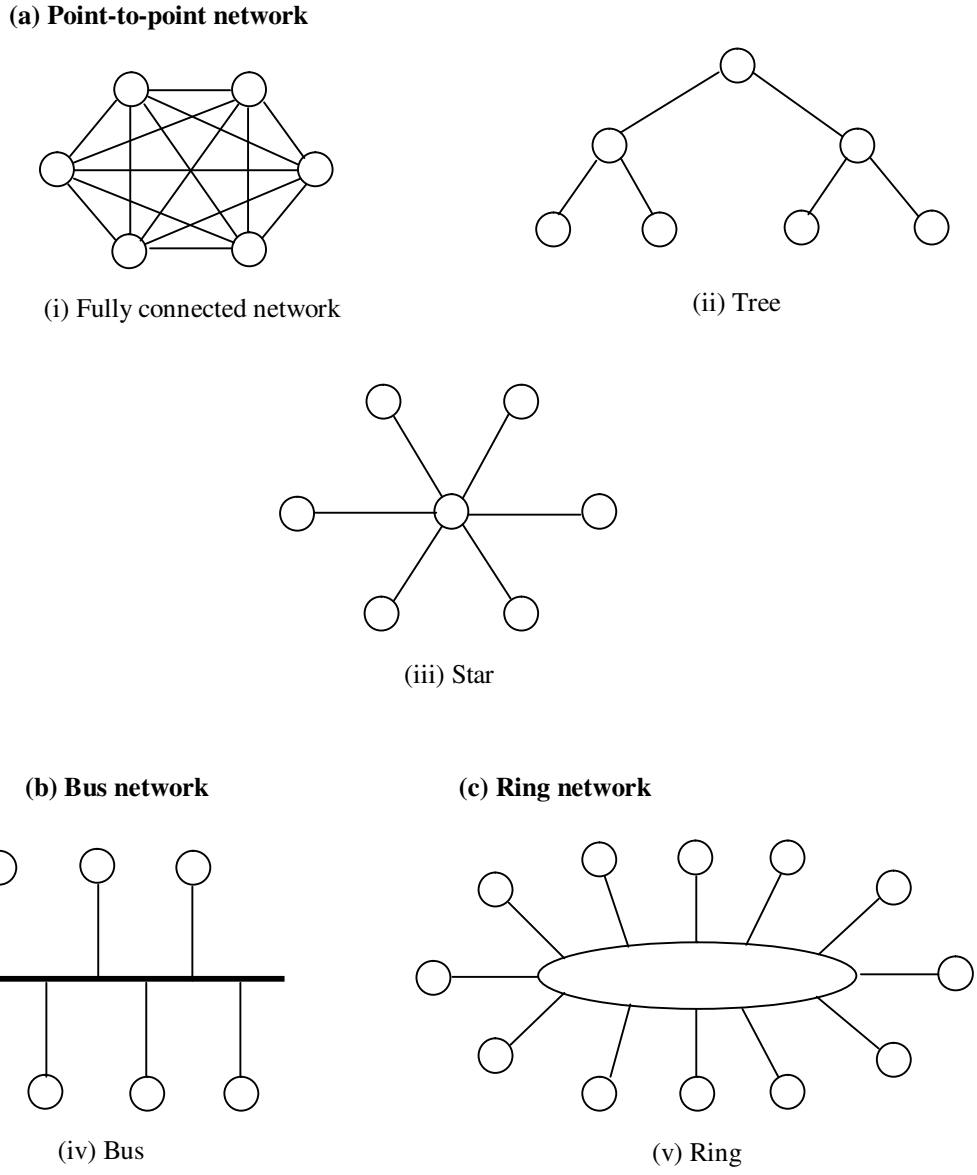


Figure 2.4 Various types of network topologies

2.5.2 Byzantine Generals Problem of Unsigned Message

Lamport, Shostak and Pease (1982) named the consensus problem for arbitrary faults as Byzantine Generals Problem (BGP) and gave two variants of the BGP according to the generals' message system. In the development of the traditional Byzantine Agreement Protocol (BAP) to solve the BGP, they were enlightened by the cryptographic papers, which were written by Diffie and Hellman (1976), and Rivest, Shamir and Adleman (1978). For both the generals' message system, i.e. the oral message and written message, they adopted the assumptions that required the application of the knowledge of cryptography.

In this section, the BGP of oral message will be discussed, whereas BGP of written message will be discussed in Section 2.5.3. *Oral message* is also known as *unsigned message* since the message is forgeable. On the other hand, *written message* is non-forgeable and restricts the ability of the faulty node from lying. This feature makes the written message to have another name called *signed message*.

For BGP with unsigned message, there are three conditions to be realised using the cryptography (Lamport et al., 1982). The conditions are listed below:

(E.1) Every message that is sent is delivered correctly.

(E.2) The receiver of a message knows who sent it.

(E.3) The absence of a message can be detected.

The first condition is in fact the access control property of the cryptography. The second condition is the authentication property. The third is the non-repudiation property. Conditions (E.1) and (E.2) prevent a faulty node from interfering with the communication between other nodes. This is because by (E.1), the faulty node cannot interfere with the message sent by the other nodes; and by (E.2), the faulty node cannot confuse the intercourse of other nodes by introducing spurious messages. Condition (E.3) prevents a faulty node from trying to foil a decision by simply not sending message.

In the context of a computer system and distributed network, conditions (E.1) and (E.2) imply that the algorithm of the traditional BAP works for processors directly connected by point-to-point links. A link failure is counted as one of the processor failures (node failures) since it is indistinguishable from a processor failure. Condition (E.3) requires the clock synchronisation of the sending nodes and the receiving nodes to some allowable time delay. It means that the maximum time for both message generation and transmission is stated and known.

For a network with n nodes, the BAP with unsigned message can handle a maximum of m faulty nodes, where $n \geq 3m+1$. In a more accurate manner, the relationship of n and m are given by Equation (2.3).

$$m = \text{floor}\,[(n-1)/3] = \lfloor (n-1)/3 \rfloor \tag{2.3}$$

2.5.3 Byzantine Generals Problem of Signed Message

For Byzantine Generals Problem of signed message, in addition to the cryptographic properties of (E.1), (E.2) and (E.3) in Section 2.5.2, its BAP solution requires one more condition. The additional condition (E.4) as stated below is a combination of two sub-conditions, i.e. (E.4i) and (E.4ii). Conditions (E.4i) and (E.4ii) request for the integrity and authentication properties of the cryptography (Lamport et al., 1982).

(E.4i) The signature of a loyal node cannot be forged. Any alteration of the contents of the signed message of the loyal node can be detected.

(E.4ii) Anyone can verify the authenticity of a node signature.

Both conditions, (E.4i) and (E.4ii), ensures that the signature of a loyal node cannot be forged. However, a traitorous node signature can be forged to another traitorous node. This creates a situation of collusion among the faulty nodes. Unlike

the BGP of unsigned message handling a maximum of *m* faulty nodes for a network of *n* nodes, BGP of signed message can cope with any number of nodes and possible faulty nodes. The minimum number of nodes for BAP of signed message is three (Lamport et al., 1982).

In the BAP of signed message, a commander node or source node delivers the signed message to each of its lieutenant nodes. Then each lieutenant node adds its signature to the received message, which is signed previously by the commander node. After the addition of the signature of the lieutenant node, the lieutenant node sends the message to the other lieutenant nodes, which will sign the message again and send it to the others, and so on. In other words, a lieutenant node must effectively receive one signed message, generate several copies of it, sign them, and subsequently send those signed copies out.

It is not important how these copies are obtained. A single message may be photocopied, or each message may consist of a stack of identical messages, which are signed and distributed upon requirement. In this thesis, the focus of study is on the ANN based BAP of unsigned message against the traditional BAP of unsigned message as in Section 2.5.2.

2.6 TRADITIONAL BYZANTINE AGREEMENT PROTOCOL (BAP)

In Section 2.5.2, the BGP of unsigned message is discussed. In this section, its solution, which is normally known as the traditional Byzantine Agreement Protocol (BAP) is presented in further details. As in Equation (2.3), for a network of *n* nodes, a maximum of *m* faulty nodes or traitorous nodes can be tolerated (Pease, Shostak &. Lamport, 1980). In addition, when the faulty status is unknown, it requires each node to exchange messages for a minimum of (*m*+1) rounds to reach a Byzantine Agreement (Fischer & Lynch, 1982). One *round* includes one cycle of message exchange among each node and the computation for making up decision. Hence the complexity of a malicious network to reach Byzantine Agreement increases as the number of nodes grows up.

To realise the model of the traditional BAP, some simple FCNs with each node ignorant about the faulty status of other nodes can be investigated. For instance,

it can be shown that it is impossible to solve the BGP with only three nodes. There are two cases for this situation. One is to have a lieutenant node as the faulty node, and the other case is to have the commander node as the faulty node. Both of these two cases are illustrated in Figure 2.5 and Figure 2.6.

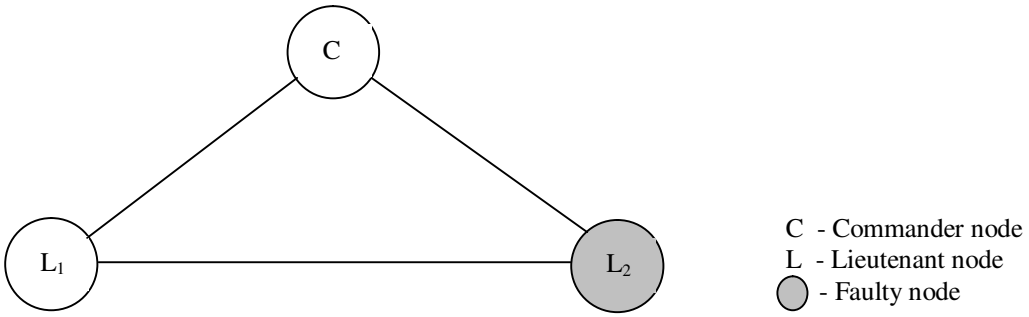


Figure 2.5 FCN-3 with lieutenant node L_2 as the faulty node

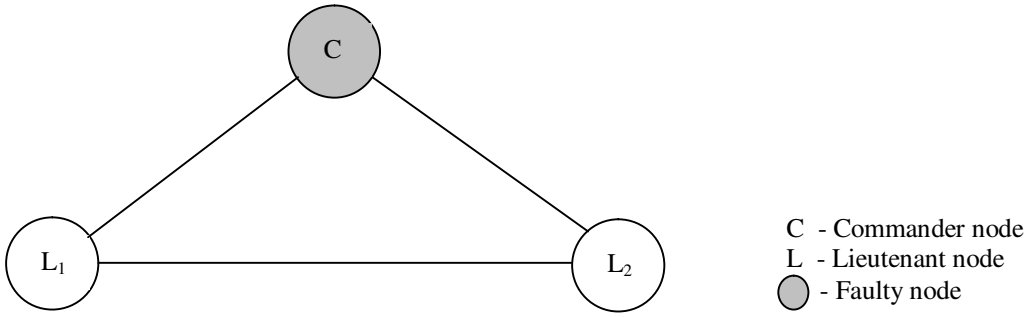


Figure 2.6 FCN-3 with commander node *C* as the faulty node

The Byzantine Generals Problem seems deceptively simple. However the difficulty is indicated by the surprising fact that if the generals can send only unsigned messages, then there is no workable solution unless more than two-third of the nodes are loyal. To be particular, for a FCN with three nodes, no solution is possible for the existence of a single faulty node. For a communication network using unsigned message or oral message, the contents of the message are completely under the control of the sender. An arbitrary faulty sending node can transmit any possible message.

With reference to the first case of the FCN-3 with one faulty node as in Figure 2.5, let us consider the decision to be made is either bit "0" or bit "1". The commander node C is a loyal node and sends a bit "1". Lieutenant node L_2 is a faulty node and delivers a message, saying that it received a bit "0", to loyal lieutenant node L_1. For the ICCs of (C.2) to be satisfied, L_1 must obey to decide on the bit "1".

In the second case as in Figure 2.6, the commander node C is traitorous. Node C sends a message of bit "1" to L_1 and a message of bit "0" to L_2. Then the loyal lieutenant node L_2 delivers its message to L_1 that it received a bit "0". Lieutenant node L_1 does not know which node is faulty, and what message the commander node actually has sent to the lieutenant node L_2. Therefore, from the point of view of L_1, both the first case and the second case appear exactly to be the same scenario to L_1. If the faulty node lies consistently, then there is no way for L_1 to distinguish between these two situations. Consequently, follows from ICCs of (C.2), L_1 must decide on the bit "1" regardless of the faulty status of the commander node. In a nutshell, whenever the L_1 receives a bit "1" from C, L_1 must accept it as its final decision.

On the other hand, as in Figure 2.6, if loyal lieutenant node L_2 receives a bit message "0" from the commander node, then L_2 must accept it even if L_1 tells it that the commander node sent a bit "1". As a result, in the scenario illustrated by Figure 2.6, L_2 must accept the bit "0" message, and L_1 must accept the bit "1" message. This situation violates the ICCs of (C.1). Hence, in the presence of a single faulty node, there exists no solution for a three-node fully connected network.

The simplest solution for BGP of unsigned message is a fully connected network (FCN) consisting of four nodes. For $n = 4$, a maximum of one faulty node can be tolerated. The faulty node can be either the commander node (source node) or the lieutenant node (receiver node). Therefore, for the network topology of FCN-4, there exist two situations having the same network architecture. These two situations of FCN-4 are shown in Figure 2.7 and Figure 2.8. Figure 2.7 shows the FCN-4 with one faulty lieutenant node, whereas Figure 2.8 shows the FCN-4 with one faulty source node. All the links are bi-directional, and it is assumed that there are no message interruption, fabrication, interception, modification and repudiation along the links.

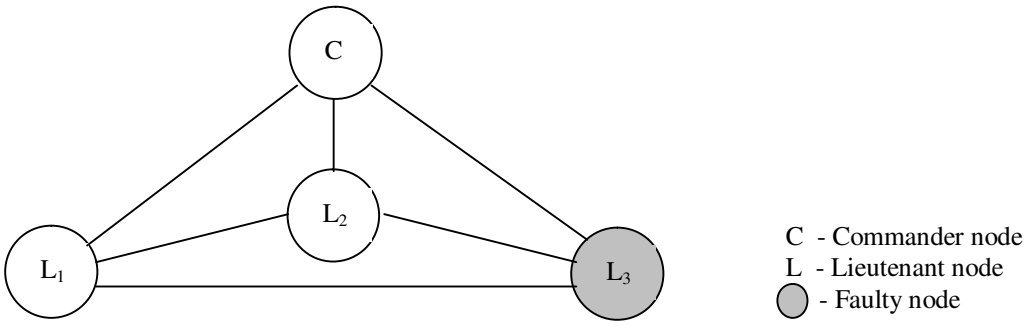


Figure 2.7 FCN-4 with lieutenant node L_3 as the faulty node

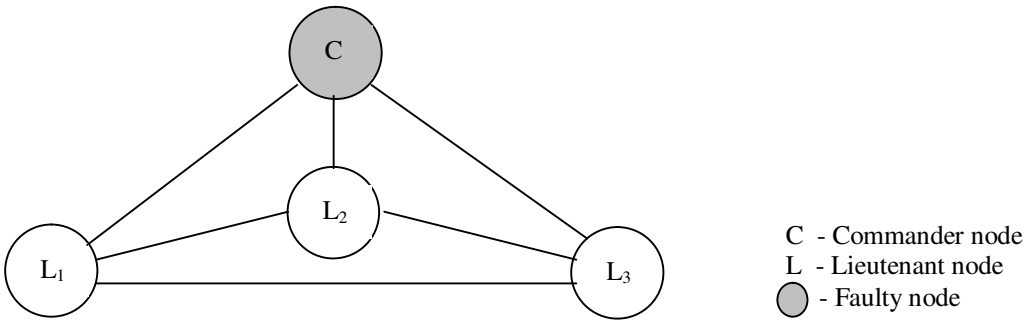


Figure 2.8 FCN-4 with commander node *C* as the faulty node

To illustrate the protocol of traditional BAP, assume a function *majority* with a property that if a majority of the values v_i equals to v, then $majority(v_1,v_2,v_3,\ldots,v_{n-1})$ equals v. When there is no majority value, a value called as DEFAULT, which is normally a pre-set value, will be adopted. The DEFAULT value can be either the bit "0" or bit "1".

In addition, assume another function *UM*, which represents the number of round in exchanging the unsigned message among the nodes. *UM(0)* represents the case of no faulty node and *UM(m)* represents the case of *m* faulty nodes. Argument *n* represents the number of nodes in a network, and argument *m* represents the number of faulty nodes. Also, assume a DEFAULT value, either a bit "1" or a bit "0", in case the faulty commander node is reluctant to send a message. For instance, if a DEFAULT value of $v_{def} = 0$ is used, the algorithm of the traditional BAP will be as follows (Lamport et al., 1982):

The algorithm:

For a fully connected network (FCN) with n nodes and m traitors ($n \geq 3m+1$), a node adopts a DEFAULT value ($v_{def} = 0$) if the faulty commander does not send a message. A function $majority(v_1,v_2,v_3,\ldots,v_i,v_{n-1}) = v$ if a majority of the values $v_i = v$.

UM(0) for m=0:

i. The commander node sends its value v to every lieutenant node.

ii. Each lieutenant node uses the value it receives from the commander, or uses the value DEFAULT ($v_{def} = 0$) if it receives no value.

UM(m) for m>0:

i. The commander node sends its value to every lieutenant node.

ii. For each lieutenant node i, let v_i be the value node i receives from the commander, or else be DEFAULT if it receives no value. Lieutenant node i acts as the commander in algorithm *UM(m-1)* to send the value v_i to each of the other *(n-2)* lieutenant nodes.

iii. For each lieutenant node i, and each $j \neq i$, let v_j be the value lieutenant node i received from lieutenant node j in step ii. via the algorithm *UM(m-1)*, or else DEFAULT if it received no such value. Lieutenant node i uses the value of $majority(v_1,v_2,\ldots,v_i,v_j,\ldots,v_{n-1})$.

To prove the theorem that algorithm of *UM(m)* solves the BGP for arbitrary m, we need to prove a lemma first before the proof of the theorem for *UM(m)*

(Lamport et al., 1982). Let us assume that they are named as *Lemma BGP* and *Theorem BGP*. Their statements and proofs are as below:

Lemma BGP: For any m and k, algorithm *UM(m)* satisfies ICCs of (C.2) if there are more than $(2k+m)$ nodes and at most k traitors.

Proof: The proof is based on the induction on m. Using (E.1), it is obvious that the trivial algorithm *UM(m)* works if the commander is loyal. Hence the lemma is true for $m = 0$ and *UM(0)* satisfies ICCs of (C.2). For $m > 0$, we assume that *UM(m-1)* is true and only the *UM(m)* is required for proof.

In step i. of the *UM(m)*, the loyal commander node sends a value v to all the other $(n-1)$ lieutenant nodes. In step ii., each loyal lieutenant node applies *UM(m-1)* with $(n-1)$ nodes.

By hypothesis, $[n > 2k+m]$ or $[n - 1 > 2k + m - 1]$. Applying the induction hypothesis, the conclusion is that every loyal lieutenant node gets $v_j = v$ from each loyal lieutenant node j.

Since there are at most k traitors and $[2k + m - 1 \geq 2k]$, we have $k < (n-1)/2$, a majority of the $(n-1)$ lieutenant nodes are loyal. Therefore, each loyal lieutenant node has $v_i = v$ for a majority of the $(n-1)$ nodes values i, and it obtains a $majority(v_1,v_2,\ldots,v_i,v_j,\ldots,v_{n-1}) = v$ in step iii. to prove the ICCs of (C.2).

Theorem BGP: For any m, algorithm *UM(m)* satisfies conditions ICCs of (C.1) and (C.2) if there are more than $3m$ nodes and at most m traitors.

Proof: The proof is based on the induction of m as in the Lemma BGP. If there are no traitors, then it is obvious to deduce from

(E.1) that *UM(0)* satisfies ICCs of (C.1) and (C.2). Then assume that the theorem is true for *UM(m-1)* and prove it for *UM(m)*, $m > 0$. There exist two cases.

For the first case, the commander node is assumed to be loyal. Taking $k = m$ from the Lemma BGP, it is found that *UM(m)* satisfies ICCs of (C.2). If the commander is loyal, then ICCs of (C.1) follows from ICCs of (C.2), so only the (C.1) is required for verification in the case that the commander node is a traitor.

Now consider the second case where the commander node is a traitor. In an n-node network, there are at most m traitors, and the commander is one of them, so a maximum of (m-1) lieutenant nodes are traitors. There are more than $3m$ general nodes, and hence more than ($3m$-1) lieutenant nodes. Since ($3m$-1) > 3(m-1), the application of induction hypothesis concludes that *UM(m-1)* satisfies the ICCs conditions of (C.1) and (C.2).

Hence for each j, any two loyal lieutenant nodes obtain the same value for v_j as in step iii. of the *UM(m)*. This conclusion follows from ICCs of (C.2) if one of the two lieutenant nodes is j; or from ICCs of (C.1) otherwise. As a result, any two lieutenant nodes get the same vector of values $v_1, v_2, v_3, \ldots, v_{n-1}$, and therefore the same value *majority*($v_1, v_2, v_3, \ldots, v_{n-1}$) in step iii. of the *UM(m)*, which proves the ICCs of (C.1).

As given the proof above, the BGP of unsigned message is solvable for an n-node network with a maximum of m faulty nodes, where $n \geq 3m + 1$. This solution is called as traditional Byzantine Agreement Protocol and the simplest solution available is a fully connected network of four nodes (FCN-4).

2.7 IMPLEMENTATION OF TRADITIONAL BAP

To understand the traditional BAP better, like many other cases, the best way is to go through some examples. In this section, two given examples are the FCN-4 and FCN-7. For the example of FCN-4, both faulty status of the commander node is presented. For the FCN-7, only the case of loyal commander node is presented.

2.7.1 4-Processor Distributed System

For a 4-processor distributed system, we can simulate it in a form of a fully connected network of four nodes (FCN-4). With a total of four nodes, only a maximum of one faulty node can be tolerated as from Equation (2.3). The faulty node can be either the commander node or one of the lieutenant nodes.

The first case where one of the lieutenant nodes is the faulty node is illustrated in Figure 2.7. As seen in the figure, the lieutenant node L_3 is assumed to be malevolent. Due to the symmetrical property of the FCN, assuming any one of the lieutenant nodes, either L_1, L_2, or L_3, as the faulty node is in fact the same.

From either the complex backward induction method (Fischer & Lynch, 1982) or the easy forward induction method (Aguilera & Toueg, 2000), it is shown that for any algorithm to assure the interactive consistency conditions (ICCs) in the presence of m faulty processors, it requires at least (m+1) rounds of communication. Hence for FCN-4 with m = 1, a minimum of two rounds of message exchanges is required to achieve the Byzantine Agreement.

For the case where L_3 is faulty as in Figure 2.7, let the commander node C to deliver its value v to every lieutenant node L_1 to L_3 in the first round of communication. Each lieutenant node receives the value from the commander node, and then acts as a commander to send its received value to the other lieutenant nodes during the second round of communication. The faulty node L_3 sends out arbitrary value x and y to the other lieutenant nodes in order to confuse them for making up the Byzantine Agreement. At the end of second round, each node has a set of received value. The majority function is applied to obtain the consensus among the loyal nodes. Figure 2.9 shows the summary of the rounds of message exchanges.

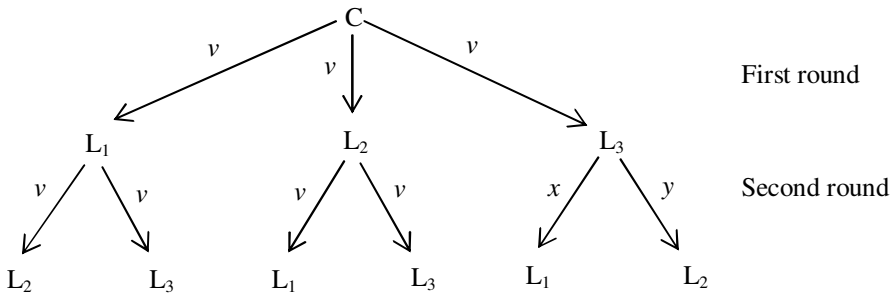


Figure 2.9 Rounds of communication for FCN-4 with $m = 1$ at L_3

At the end of the first round of communication, at

L_1 : $v_1 = v$

L_2 : $v_2 = v$

L_3 : $v_3 = v$

At the end of the second round of communication, at

L_1 : $v_1 = v, v_2 = v, v_3 = x$

Decision = majority(v, v, x) = v

L_2 : $v_1 = v, v_2 = v, v_3 = y$

Decision = majority(v, v, y) = v

L_3 : $v_1 = v, v_2 = v, v_3 = x$

Decision = majority(v, v, v) = v

Byzantine Agreement = v

At the end of the second round, each lieutenant node has received a set of values and arrives at the same decision to fulfil the ICCs of (C.1). The decision is the majority of a set of values. For this case (FCN-4 with faulty L_3), all the loyal lieutenant node decisions are the same as the value sent by the commander node C, and hence the ICCs of (C.2) is satisfied.

For the second case as in Figure 2.8 (FCN-4 with faulty *C*), the number of rounds of communication is the same with the case FCN-4 with faulty node L_3. However, this time we have a faulty commander node instead of a loyal commander node. The summary of the first round and second round of message exchanges are illustrated as in Figure 2.10. At the first round, *C* sends arbitrary values *x* and *y* to L_1 and L_2 respectively, and is reluctant to deliver any value to L_3.

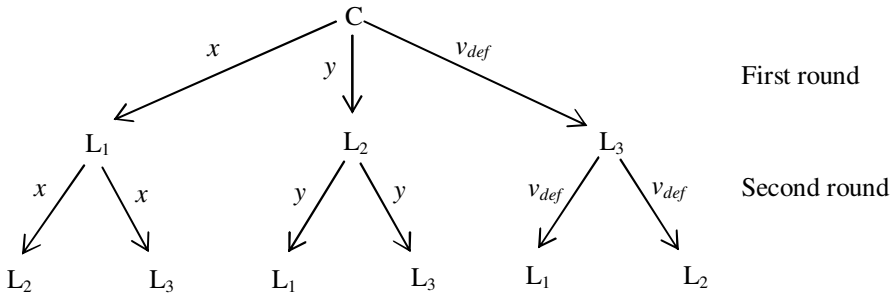


Figure 2.10 Rounds of communication for FCN-4 with $m = 1$ at *C*

At the end of the first round of communication, at

L_1 : $v_1 = x$

L_2 : $v_2 = y$

L_3 : $v_3 = v_{def}$

At the end of the second round of communication, at

L_1 : $v_1 = x$, $v_2 = y$, $v_3 = v_{def}$

Decision = majority(x, y, v_{def})

L_2 : $v_1 = x$, $v_2 = y$, $v_3 = v_{def}$

Decision = majority(x, y, v_{def})

L_3 : $v_1 = x$, $v_2 = y$, $v_3 = v_{def}$

Decision = majority(x, y, v_{def})

The three loyal lieutenant nodes receive the same value majority(x, y, v_{def}) regardless of whether or not any of the three values x, y, and v_{def} are equal. Therefore their Byzantine Agreements are the same, i.e. majority(x, y, v_{def}), and hence the ICCs of (C.1) and (C.2) are respected.

2.7.2 7-Processor Distributed System

For FCN-4, FCN-5 and FCN-6, it can be proven form Equation (2.3) that all the three FCN can tolerate a maximum value of m = 1. Hence the three of them are having the same rounds of communication. Meanwhile, for FCN-7, FCN-8 and FCN-9, the m can be 0, 1 or 2. The rounds of communication have to be at least three to fulfil the ICCs and to achieve the Byzantine Agreement among all the loyal nodes. In this section we study the FCN-7 which shows three rounds of message exchanges. Figure 2.11 shows the network topology of FCN-7 with L_5 and L_6 as the faulty nodes.

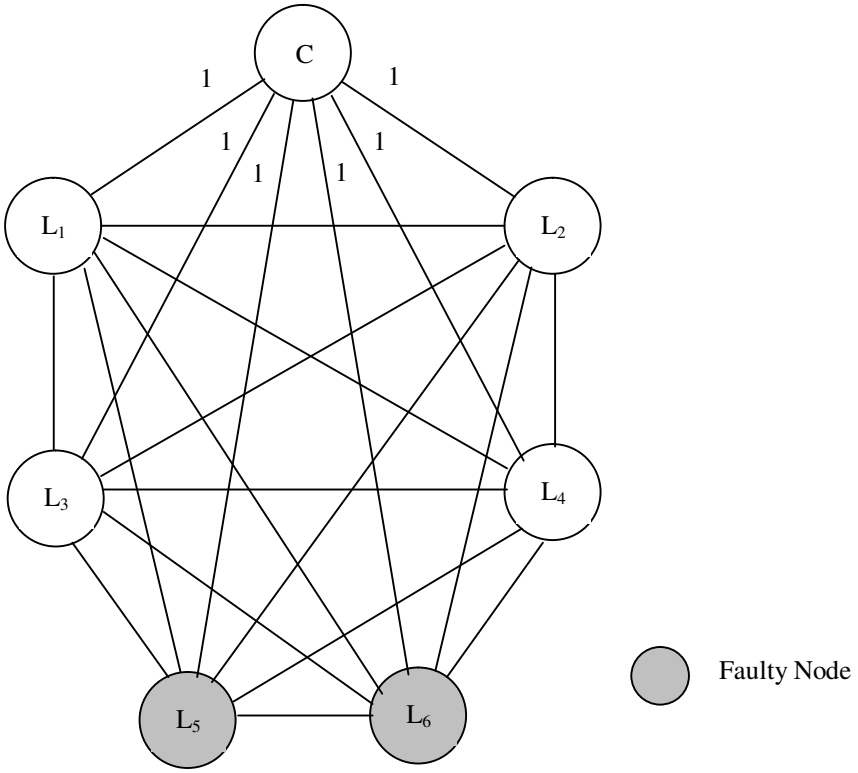


Figure 2.11 FCN-7 with first round of message exchange

In Figure 2.11, the first round of message exchange is also shown, where the loyal commander node *C* sends bit "1" to every lieutenant node. In the second round, every lieutenant node sends its value to all the other lieutenant nodes. The messages received by each lieutenant node are shown in Table 2.1. Lieutenant nodes L_5 and L_6 are both traitorous, sending out arbitrary messages and recording arbitrary messages received from the other lieutenant nodes.

Table 2.1 Second round of message exchange

Sender Node	Received message at each receiver node					
	L_1	L_2	L_3	L_4	L_5	L_6
L_1	1	1	1	1	0	0
L_2	1	1	1	1	0	0
L_3	1	1	1	1	1	1
L_4	1	1	1	1	0	1
L_5	1	0	0	0	1	0
L_6	0	0	1	0	0	0

* L_5 and L_6 are faulty node

To understand the Table 2.1, the receiver node L_1 is taken as an example. L_1 receives bit "1" from commander node *C*. This bit "1" is indicated by the bit "1" at the coordinate of (L_1, L_1). The other bits received by node L_1 are "1" from L_2, "1" from L_3, "1" form L_4, "1" from L_5, and "0" from L_6. To make it clear, it is as below.

(L_1,L_1) = bit "1" from *C* (L_2,L_1) = bit "1" from L_2

(L_3,L_1) = bit "1" from L_3 (L_4,L_1) = bit "1" from L_4

(L_5,L_1) = bit "1" from L_5 (L_6,L_1) = bit "0" from L_6

Subsequently, at the third round of communication of FCN-7, which is the final round, each loyal node is obligated to decide the Byzantine Agreement independently by using the majority function over the set of values it received from the other nodes. Table 2.2 shows the message received by all the loyal lieutenant nodes, i.e. L_1, L_2, L_3, and L_4, during the third round of message exchange. Note that only the coordinates (L_x,L_y) where $\{x \,\&\, y = 1, 2, 3, 4\}$ will hold the loyal bit "1". The other coordinates will be affected by the faulty nodes to hold either bit "0" or bit "1". For instance, in the lieutenant node L_1, the messages received are loyal bit "1" at (L_x,L_y) where $\{x \,\&\, y = 1, 2, 3, 4\}$. The other coordinates will hold a faulty bit "0".

Table 2.2 Message received by each loyal lieutenant node after the third round of message exchange, and the Byzantine Agreement at each node

Message received by L_1						Local	Node
L_1	L_2	L_3	L_4	L_5	L_6	MAJ	MAJ
1	1	1	1	0	1	1	1
1	1	1	1	1	0	1	
1	1	1	1	0	1	1	
1	1	1	1	1	1	1	
0	0	1	0	0	0	0	
0	0	1	1	1	0	DEF	

Message received by L_2						Local	Node
L_1	L_2	L_3	L_4	L_5	L_6	MAJ	MAJ
1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	
1	1	1	1	0	0	1	
1	1	1	1	1	0	1	
1	0	0	1	0	0	0	
1	1	0	0	0	0	0	

Message received by L_3						Local MAJ	Node MAJ
L_1	L_2	L_3	L_4	L_5	L_6		
1	1	1	1	0	0	1	1
1	1	1	1	0	1	1	
1	1	1	1	0	1	1	
1	1	1	1	1	1	1	
0	0	1	0	1	0	0	
0	0	0	1	1	0	0	

Message received by L_4						Local MAJ	Node MAJ
L_1	L_2	L_3	L_4	L_5	L_6		
1	1	1	1	0	0	1	1
1	1	1	1	0	1	1	
1	1	1	1	1	1	1	
1	1	1	1	0	0	1	
0	1	0	1	1	0	DEF	
0	0	1	0	0	1	0	

*Local MAJ : Local majority (determine value held by each sender)

*Node MAJ : Node majority (determine Byzantine Agreement at each node)

*DEF : Pre-set DEFAULT value before operation of *UM(m)*

From Table 2.2, we can see that all the loyal lieutenant nodes (L_1, L_2, L_3, and L_4) reach the same node majority. This means all the loyal lieutenant nodes achieve the same Byzantine Agreement, and hence the ICCs of (C.2) is satisfied. Since the Byzantine Agreement is the same as the value delivered by the source or the commander node, the ICCs of (C.1) is also fulfilled. To sum up, the BGP of FCN-7 is solved by the traditional BAP.

2.8 COMPLEXITY OF THE TRADITIONAL BAP

To analyze the efficiency of any algorithm, the notation of Big-O is always used. Big-O, which is symbolised as *O(f(n))* with *f(n)* as a function of the argument *n* of the algorithm, represents the order of magnitude of the computational complexity of an algorithm. The argument *n* is always the size of input to an algorithm. This field of study is known as the *complexity theory* in the world of mathematics. The computational complexity of an algorithm is normally measured in the variables of time complexity *T* and space complexity *S*. Both of the variables are then expressed in the Big-O notation.

For the traditional BAP as stated in the algorithm of *UM(m)*, its complexity can be known by first calculating the total number of messages that are being exchanged. It is well known that at least (*m*+1) rounds of communication are required for the loyal nodes to achieve the Byzantine Agreement in unison.

Assume an FCN-*n* with *m* faulty nodes, then we have the number of messages being communicated at each round as below:

First round	:	(*n*-1) messages
Second round	:	(*n*-1)(*n*-2) messages
Third round	:	(*n*-1)(*n*-2)(*n*-3) messages
.	.	.
.	.	.
.	.	.
*m*th round	:	(*n*-1)(*n*-2)(*n*-3)…(*n*-*m*) messages
(*m*+1) round	:	(*n*-1)(*n*-2)(*n*-3)…(*n*-*m*)(*n*-(*m*+*1*)) messages

By taking the sum of messages during all the rounds of communication, we have the total messages being exchanged in the traditional BAP of an *n*-node network with *m* faulty nodes in term of $Msg_{Trad.}$ as in Equation (2.4) below:

$$Msg_{Trad.} = n - 1 + \sum_{k=1}^{m} \frac{(n-1)!}{(n-k-2)!} \tag{2.4}$$

From Equation (2.4), we can derive the complexity of algorithm *UM(m)* in Big-O notation as $O(n^{m+1})$. Since m is not a constant but a function of n as well, where $m = \lfloor (n-1)/3 \rfloor$, the *UM(m)* with $O(n^{m+1})$ belongs to a class of algorithm with exponential complexity. Therefore, we can conclude our complexity analysis that traditional BAP is an exponential-time algorithm.

From the analytical investigations, we obtain Figure 2.12 to show the graph of total message exchanges, $Msg_{Trad.}$, versus the number of nodes, n, with maximum allowable m. Y-axis shows the total message exchanges in the scale of common logarithm 10. It can be seen in the figure clearly that the complexity of the BGP jumps for every increment of the number of faulty nodes, m.

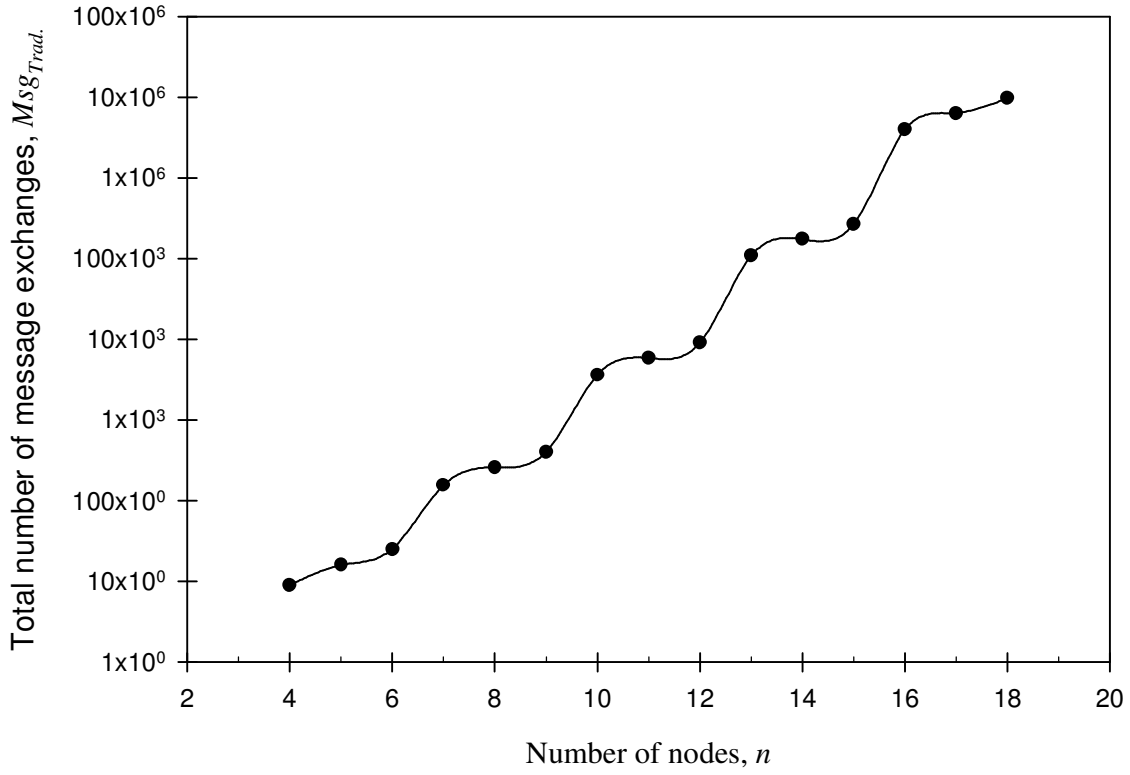


Figure 2.12 Total number of message exchanges, $Msg_{Trad.}$, versus number of nodes, n, for traditional BAP

For FCN-4, FCN-5 and FCN-6, they are all having a value of m = 1, which form the lowest group. Then it is the group of FCN-7, FCN-8 and FCN-9, which is a group of m = 2. The third group is FCN-10, FCN-11 and FCN-12 having m = 3. Then the following group is FCN-13, FCN-14 and FCN-15. The highest group in Figure 2.12 consists of FCN-16, FCN-17 and FCN-18 with m = 5. The significant jump of total messages between each group is due to the increment of one round of communication between each group. From the case of FCN-4 with a total of 9 messages being exchanged to the FCN-18 with 9714769 messages, we can see the reason why traditional BAP is an exponential-time algorithm.

2.9 CONCLUSIONS

In this chapter, the topic of dependability of a distributed system has been presented in Section 2.2. The major concerns of dependability are reliability and availability. In addition, the dependability attributes, impairments, measures and means are discussed. Then, in Section 2.3, all the possible faults within a system are classified according to the viewpoint of pattern of occurrence and viewpoint of origin of fault. Here the concept of universal fault class, which is well known as the Byzantine fault, is brought out. The Byzantine fault is then further extended to the main issue of this thesis: The Byzantine Generals Problem. Before this, the fault tolerance techniques, i.e. masking redundancy and dynamic redundancy, are briefly presented in Section 2.4.

In Section 2.5, the in-and-out of the Byzantine Generals Problem (BGP) is discussed. BGP was firstly solved by Lamport, Shostak and Pease in 1982. Both of the generals' message systems, i.e. oral message and written message, are included in this section. The solution of the BGP is proven concisely in Section 2.7. In this thesis, the algorithm for the BGP solution is called as traditional Byzantine Agreement Protocol (BAP). This is to ease the later discussions in the coming chapters when the traditional BAP is used to compare with the proposed BAP in this thesis. In the context of both of the research project and this thesis, the proposed BAP is called as artificial neural network based Byzantine Agreement Protocol (ANN based BAP). The ANN based BAP is documented in Chapter 4.

In Section 2.7, two examples of the traditional BAP are presented. These two examples are based on two types of fully connected networks (FCN), which are the FCN-4 and FCN-7. For FCN-4, both types of the faulty status of the commander node are considered, i.e. the case of faulty commander node and the case of loyal commander node. Meanwhile, for FCN-7, only the case of loyal commander node is discussed. In Section 2.8, the algorithm of the traditional BAP is analyzed to determine its computational complexity.

CHAPTER 3: A NEW APPROACH TO BYZANTINE AGREEMENT PROTOCOL

3.1 INTRODUCTION

In Chapter 2, the first Byzantine Agreement Protocol (BAP) proposed by Lamport, Shostak and Pease (1982) is discussed in details. The BAP is important in designing a fault-tolerant distributed system. BAP helps the loyal processors to reach consensus in the presence of faults. Since the traditional BAP in 1982, many BAPs have been proposed to solve a variety of Byzantine Generals Problem (BGP).

These BAPs differ from each other due to the different assumptions on network topology, processor fault and link fault. Among so many types of network topology, there are point-to-point networks, bus network and ring network as in Figure 2.4. Nevertheless, we can have another way of classification based on the patterns of message delivery. We can divide the network topologies into fully connected network, non-fully connected network and broadcast network. For processor faults, there are Byzantine fault (arbitrary fault), dormant fault and hybrid fault at the node itself. Dormant fault is a subset of Byzantine fault such as timing fault and omission fault. Hybrid fault is a mixed fault with more than one kind of faults. Meanwhile, the link faults are arbitrary fault and hybrid fault in the path between two nodes. These BAPs are listed in Table 1.1.

Early BAPs are normally based on FCN model with a single universal fault namely Byzantine fault. However, designing a system to tolerate only Byzantine fault is of high expense in terms of space, time and computation. Therefore system with lower reliability is designed based on higher order of fault class such as dormant fault and link fault. Wang and Kao (2001) proposed a new approach to BGP in brief. They adopted the artificial neural networks (ANN) to ease the full expense of BGP in the requirements for space, time and computation.

In this chapter, further investigation is done on the artificial neural network based Byzantine Agreement Protocol (ANN based BAP). A more thorough series of

experiments are carried out. In addition, the ANN based BAP is modified and improved to reduce the rounds of message exchanges.

3.2 ARTIFICIAL NEURAL NETWORKS

Artificial neural network (ANN) was defined by Gurney (1997) as an interconnected assembly of simple processing elements, i.e. *units* or *nodes*, whose functionality was loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or *weights*, obtained by a process of adaptation to, or *learning* from, a set of training patterns. Haykin (1994) mentioned that neural network was a massively parallel distributed processor resembling the brain in two respects as below.

i. Knowledge is acquired by a network through learning process.

ii. Inter-neuron connection strengths known as synaptic weights are used to store the knowledge.

Paving through the development of artificial neural networks (ANN), one can notice that ANN is one of the products of artificial intelligence (AI). In Fu (1994), AI was defined as the study of intelligence behaviour that concerned with the implementation of a computer program exhibiting intelligent behaviour. From the computational point of view, there are two ways where the intelligence emerges, i.e. symbolism and connectionism. The first approach of symbolism models intelligence using symbols, rules and equations. The second approach of connectionism makes use of the connections and associated weights.

Historically, computing takes a variety of forms. Since 1940s, symbolism takes the lead dominating the computing world via the programmed computing or traditional computing (Schalkoff, 1997). The traditional AI or ruled based AI is in fact simply expressed on a high level of traditional computing (Swingler, 1996). It is a technique of constructing algorithms from a set of simple structures such as *if … then*, *while … then*, and *repeat … then*.

Its advantage is that the rules and functions may be formally proven and stated. Besides, if the rules are known, it is simple to add the rules to an expert system. Its disadvantage is that it is an intensive labour task and is sometimes impossible to extract rules from a human expert or to construct an algorithm from the available programming structures. In other words, it means the weakness occurs when the rules cannot be clearly defined and stated into functions and equations. This is the reason why the traditional AI is known as logic-oriented approach to intelligence. Those programs, such as *LISP* and *PROLOG*, are designed on traditional AI technique to process logical predicates. Many years pass, fuzzy logic is then invented to allow the existence of less precise rules. However, the fuzzy concept derives those values like *hot*, *happy* and *healthy* simply from design rather than from a statistical process. For complex system, it is still a failure to build up a general model by using the logic-oriented approach.

In 1970s, there is deeper understanding on intelligent behaviour. It is discovered that a computer program can manage to display intelligent behaviour if the domain it deals with is sufficiently narrowed. This discovery has been the outcome of the transformation of AI from the logical-oriented approach towards the knowledge-based approach. The success of the historical heuristic programming projects namely DENDRAL (1965-1983) and MYCIN (1970s) has close link to this transformation (Buchanan & Shortliffe, 1984). DENDRAL is an acronym for "Dendritic Algorithm". DENDRAL is applied in the area of organic chemistry and MYCIN is applied for medical diagnosis. Both of these applications are two of the earliest expert systems.

From the development of DENDRAL and MYCIN, it is learnt that to make a system intelligent, it is the *knowledge* rather than the *inference mechanism* from the rules. Subsequently, the symbolic approach is then having a partner called neural network approach. People start to investigate the adoption of brain metaphor. The *brain metaphor* suggests that intelligence emerges through a large number of connected processing elements. The elements perform simple computations and have links among each other. Each link has its specific weight. These weights are in fact the storage of long term knowledge in a neural network. The elements are then well known as neurons or nodes.

Because of the interconnected element capability of storing knowledge in the weights, the neural network earns the name of *connectionist* (Feldman & Ballard, 1982), which is an approach of modelling intelligence described earlier. In addition to the reasoning of symbolic systems, a connectionist system of rule-based reasoning offers more in term of common sense reasoning. Later, the connectionist systems are called as parallel distributed processing systems (PDP), and then the currently famous approach of artificial neural networks (ANN) in a variety of disciplines (Bose & Liang, 1996).

3.2.1 Background of ANN

The brain is a highly complex, non-linear and parallel computer. It has the capability of performing the computations like pattern recognition and motor control many times faster than the digital computer. In 1991, Faggin pointed out that the energetic efficiency of a brain is about 10^{-16} joules per operation per second, whereas the best computer then is about 10^{-6} joules per operation per second. This is because the brain is not only massively distributed and parallel in computation, but it learns and replaces a priori program development. Due to these facts and the progress of neurobiology, McCulloch and Pitts (1943) built the first mathematical model of computing neuron. In 1949, Hebb proposed a scheme to enable neuron's connections in storing and learning knowledge. In 1954, Farley and Clark set up a random network model for adaptive stimulus-response relations. In 1958, Rosenblatt at Cornell University proposed the learning algorithm for a perceptron (a two-layer feed forward network). In 1960, Widrow and Hoff introduced an important variation of perceptron learning rule known as the Widrow-Hoff rule. Later the rule is called as the least mean square (LMS) algorithm.

ANN came to an eclipse for almost two decades when Minsky and Papert (1969) pointed out the theoretical limitations of single layer neural network in their landmark book called *Perceptrons*. The problem of designing multi-layer neural network has come to a threshold. The ANN development has slowed down during this period. Few important works are the psychological models by Anderson,

Silverstein, Ritz, Jones (1977) and Grossberg (1980), and the associated memory models by Kohonen (1977).

It is only until the significant work by Hopfield (1982) that the neural networks are resurrected. Hopfield (1982) introduced the idea of energy minimisation from physics into neural networks. The energy function formulates a new way of computation by recurrent network with symmetric synaptic connections. This kind of neural network with feedback is then named as Hopfield network. Unlike other neural network learning techniques, Hopfield network depends on a single shot learning, the learning algorithm sums the auto correlation for each pattern to become the weight matrix. It may not yet be a very realistic model of neurobiological system, but its principle allows information storage in dynamically stable networks.

In the same year, Feldman and Ballard (1982) made the term of *connectionist* renowned. Inspired by neural networks, connectionism, which is a subsymbolic process, has become the study of cognitive and AI systems (Smolensky, 1988). In contrast to symbolic AI, connectionism emphasizes on the capability of learning and discovering representations. Connectionism is then a common ground between traditional AI and ANN. Instead of the name *connectionist systems*, ANN is then to earn its third name *parallel distributed processing systems (PDP)*. The name of PDP originates from a book *Parallel Distributed Processing* written by Rumelhart and McClelland (1986). PDP has generated great impacts on computer, cognitive and biological sciences.

In 1974, Werbos suggested the idea of back propagation. Independently, Parker (1982) came to the same idea in his invention report at Stanford University. Shattering the curse on perceptrons, Rumelhart, Hinton and Williams (1986) developed the gradient descent based back propagation learning algorithm as a powerful solution to the training problem of a multi-layer neural network. NETtalk is one of the successful implementation of ANN with back propagation neural network (BPNN) as the learning algorithm (Sejnowski & Rosenberg, 1987). NETtalk is a system converting the English text into highly intelligible speech.

Since then, both the symbolic approach and neural network approach coexist in the field of AI. Symbolic approach favours the high-level cognitive reasoning ability; whereas neural network approach offers the low-level pattern recognition capability. There is no concern on which approach will take the lead. Kandel and Langholz (1992) favoured hybrid approach of symbolism and connectionism since both were complement to one another. In Table 1.1, all the BAPs are of symbolic approach except the Wang and Kao's BAP (2001), which makes use of the neural network approach.

3.2.2 Applications of ANN

ANN approach has allowed the emergence of neural computing for a large number of applications. This is because the emulation of biological computational structures may yield superior computational paradigms for some classes of problems, especially the class of NP-hard problems. These problems normally have the features like a high-dimensional problem space, complex or mathematically intractable interactions between problem variables, an empty solution space, a unique solution, problems with a set of useful solutions, and human sensory problems.

For instance, NP-class problems include labelling problems, scheduling problems, search problems, constraint satisfaction problems, pattern recognition problems, and problems handling missing, contradicting, fuzzy, malicious and probabilistic data (with random faults). To be clear, ANN is normally applied in these fields: image processing, signal processing, pattern recognition, medicine, military systems, financial systems, planning, control and search, and power systems.

Examples of image processing are image matching, pre-processing, segmentation and analysis. It also includes computer vision for circuit board inspection, image compression, stereo vision, and processing of time-varying images. For signal processing, there are seismic signal analysis, cosmic ray analysis, CDMA and morphology.

Pattern recognition is related to feature extraction, sonar and radar signal classification and analysis, speech recognition and understanding, fingerprint and iris identification, character recognition, and handwriting analysis. For medical

application, there are electrocardiogram (ECG), electromyogram (EMG) and electroencephalogram (EEG) signal analysis, diagnosis of various diseases, and medical image processing (Simon & Eswaran, 1997).

For military applications, there are undersea mine detection, radar clutter classification and tactical speaker recognition. On the other hand, financial systems apply ANN in stock market analysis, real estate valuation, credit card authorisation and securities trading. Meanwhile, in the area of planning, control and search, ANN is used for parallel implementation of *constraint satisfaction problems* (CSP), solutions to Travelling Salesman, robotics, and automation. In power systems, ANN is used for system state estimation, fault diagnosis (Bretas & Hadjsaid, 2001), transient detection and classification, fault detection and recovery (Snider & Lynch, 1997), load forecasting, and security assessment.

3.2.3 Advantages of ANN

For neural computing, there are three basic advantages over the others. The first one is its natural ability of storing the information and knowledge within the weights among the interconnected nodes. The second is its massive parallel distributed computing structures. The last one is its learning capability and generalization. Generalization means the neural network produces reasonable outputs for inputs not encountered during training. The ANN universal approximation abilities help point out its capability to solve the BAP problem. The three of these capabilities enable the neural networks to solve the currently intractable complex problems. Based on these three neural inherent properties, we can have another nine useful properties and capabilities from the ANN approach. They are the non-linearity, input-output mapping, adaptivity, evidential response, contextual information, fault tolerance, VLSI implementability, uniformity of analysis and design, and neurobiological analogy. All of the advantages will be discussed one after another.

Non-linearity: A neuron is basically a non-linear device. Hence the interconnection of neurons is by itself also non-linear. The ANN formed from the interconnection is of distributed nature. Therefore, the non-linearity of ANN is

distributed throughout the whole network. In dealing with inherently non-linear problems, the naturally distributed non-linear ANN is extraordinary useful.

Input-Output Mapping: This is an ANN property when the popular supervised learning is used to train the weights. For *supervised learning*, before the training starts, a set of labelled training samples are prepared. In the samples, each unique input signal is associated with a particular desired response. During the training process, the weights are modified to minimize the difference between the desired response and the actual response of the ANN to the input signals. The training process of the network will be repeated until the pre-set statistical criterion is reached. When the statistical criterion is satisfied, the neural network is said to have achieved a steady state and there will be no more significant change to the weights. Besides, the previously applied training samples can be used again for training by simply changing the order of the samples. This way of training is in fact from a branch of statistics dealing with model-free estimation called the study of *non-parametric statistical inference*. From the biological viewpoint, non-parametric statistical inference is known as *tabula rasa learning* (Geman, Bienenstock & Doursat, 1992). This ANN matching property is useful for pattern recognition problems, signal processing problems and control application problems. For pattern recognition, the input data to be dealt with in general are images, speeches and handwritings.

Adaptivity: This is an ANN built-in capability to adapt their synaptic weights to the environment changes. A neural network trained for a particular environment can be re-trained to handle the minor variations of the operating environmental conditions. This adaptivity property is useful in dealing with problems having the statistical situation changes with time. This time dependent circumstance is called as *non-stationary environment*. To survive in the non-stationary environment, the ANN can be designed to have the weights change in real time. The natural ANN property for input-output mapping coupled with the adaptive capability, enables it to be an ideal tool for application in adaptive pattern classification, adaptive signal processing, and adaptive control. In general, the more adaptive a system is being designed, the more robust a stable adaptive system is in performance. Adaptivity and robustness are good in noise protection and immunization.

Evidence Response: This useful property is for pattern recognition problems. ANN can be designed to provide information for selecting a particular pattern. In addition, ANN gives *confidence* in the decision made. This second information helps to reject ambiguous patterns if they arise. As a result, the evidence response of ANN improves the classification performance of the network.

Contextual Information: ANN stores the knowledge in its structural architecture and neuron activation state. Each neuron in a network is potentially affected by the global activity of all the other neurons within the same network. Therefore, it is ANN natural ability to deal with contextual information.

Fault Tolerance: Because of the natural distributed architecture, ANN is inherently having the fault-tolerant property. Implemented in hardware form, ANN performance is degraded gracefully under adverse operating conditions (Bolt, 1992). For instance, if a neuron or weight is down, the recall of a stored pattern is impaired in quality. Nevertheless, the impairment is not obvious due to the ANN distributed nature in information storage. The overall network response is only degraded seriously when the damage becomes extensive. Summing up the paragraph, it means that ANN performance shows a graceful degradation rather than an immediate catastrophic failure.

VLSI Implementability: ANN has three basic features, i.e. parallel nature, distributed nature and learning nature. Among these three, the ANN massive parallel nature makes it potentially fast in computation. This property also makes ANN ideally suited for the implementation of *very large scale integration (VLSI)* technology. VLSI provides a method to capture truly complex behaviour in a highly hierarchical form (Mead & Conway, 1980). This makes the ANN to be a possible tool for real-time applications involving pattern recognition, signal processing and control.

Uniformity of Analysis and Design: ANN uses the same notation in all the domains involving its application. We also call this feature as domain free modelling. This contributes to the neuron universality feature of being an information processor. The universality feature has its manifestations in different ways. Neurons represent an ingredient common to all neural networks. The commonality permits the ANN

possibility of sharing theories and learning algorithms in different applications of neural networks. Modular networks can be built through a seamless integration of modules.

Neurobiological Analogy: ANN originates from the simulation of brain functions. A brain is not only a physically living example of fault-tolerant parallel processor, but also fast and powerful computer. Neurobiologists use ANN as a research tool for interpreting the neurobiological phenomena. Electrical engineers discover neural applications in medical signal processing and control theory for muscle movement. Computer engineers find the hardware potential for implementing the neural network approach efficiently, especially robotics and automation. Computer scientists adopt the ANN for difficult problems to simulate the brain functions, such like optimisation, artificial intelligence, pattern recognition and data classification. Applied mathematicians make the ANN a powerful tool for modelling problems for which the explicit form of the relationships among certain variables is not known, such as the NP-hard class of problems like genome mapping.

3.3 INTEGRATION OF ANN TO BYZANTINE AGREMENT PROTOCOL

For traditional BAP, it is a logic-oriented approach of solving the BGP. It lacks all the ANN advantages mentioned above. On the contrary, the ANN based BAP, which is implemented in this thesis, includes all the ANN benefits. This has solved the time consuming problem of traditional BGP in rounds of communication for message exchange. Furthermore, ANN is inherently fault-tolerant to malicious attack because of its distributed nature. Therefore, ANN based BAP has extra fault-tolerant capability over the traditional BAP.

The integration of ANN to the BGP can be easily implemented because of the similarity of fully connected network (FCN) and artificial neural network (ANN). The symmetry of FCN and ANN allows mutual representation via the analogy approach. FCN is in fact a fully connected undirected graph having the vertices as the nodes and edges as the communication links. Each edge or link in the FCN allows the message delivery among two vertices in both directions. Contrary to FCN,

ANN being applied here is a symmetrical multi-layer perceptron. The number of layers depends on the size of a distributed network. For small distributed system, two-layer perceptron is sufficient in solving the BGP. The type of perceptron or ANN used here is a feed forward network with gradient descent based back propagation training algorithm.

For a FCN of n nodes, there will be an embedded ANN in every node of the FCN. The embedded ANN will be having (n-1) input neurons at its input layer. At the output layer, there will be three output neurons. One output neuron represents the Byzantine Agreement (BA) of binary bit "0". Another output neuron represents the BA of binary bit "1". The last output neuron represents the value of "DEFAULT" situation where no message is received or no majority value is achieved. The value "DEFAULT" is set as bit "0" or bit "1" prior to BAP execution.

For the embedded ANN, if it is a two-layer perceptron, then we will have the number of hidden neurons in the hidden layer to be half the total of input neurons and output neurons. As long as the ANN is efficient, in terms of time and epochs, to represent the non-linearity of BGP, this criterion is used to ease the design of neural network architecture (Hagan & Demuth, 1995).

Due to the fact of embedding one ANN into every node of the distributed network, a lower level of neural network is formed among all the embedded ANN through the fully connection of the distributed system. Since the embedded ANN are distributed throughout the FCN, the lower level neural network formed across the system is called as the distributed artificial neural networks (DANN). DANN is in fact a subfield of distributed artificial intelligence (DAI) (O'Hare & Jennings, 1996). On the other hand, DAI is a subfield of AI concerned with distributing and coordinating knowledge and actions in multiple agent environments. Therefore, DANN is inherently distributed and hence increases again the fault-tolerant capability of the ANN based BAP.

There are basically two main research fields in DAI: distributed problem solving (DPS) and multi-agent systems (MAS). DPS considers how to divide the task of solving a problem among a number of nodes, which cooperate in dividing and sharing knowledge about the problem and about its evolving solutions. In DPS, all

interaction strategies are integrated parts of a system. MAS considers the behaviour of a collection of autonomous agents targeting at solving a given problem. MAS is a loosely-coupled network of agents (problem solvers) working together on problems beyond their individual capabilities. For ANN based BAP proposed in this thesis to solve the Byzantine Generals Problem, it is a type of MAS.

Despite the ANN benefits, it is also important to know about its weakness before going to the next chapter on ANN based BAP. There are basically four. It is hard to explain the neural results explicitly. The knowledge representation is vague and hard to understand. Neural architecture configuration and neural network training are time consuming. Lastly, it is hard to know whether a learning algorithm converges to an optimal solution, and whether it is the most efficient.

3.4 CONCLUSIONS

Vis-à-vis the BAP listed in Table 1.1, the types of network topology, processor faults and link faults are discussed briefly in this chapter. Then the new approach to BAP (Wang & Kao, 2001) using the artificial neural networks (ANN) is introduced. All the previous BAPs before Wang and Kao's BAP are of logic-oriented approach. On the contrary, Wang and Kao's BAP is a knowledge-based approach.

In Section 3.2, ANN is documented in depth. Section 3.2.1 presents the development of ANN from its first model by McCulloch and Pitts (1943) to the ANN with BPNN by Sejnowski and Rosenberg (1987). The evolutionary story of AI and ANN is presented around the symbolism and connectionism. In Section 3.2.2, the applications of ANN are brought out. The discussed applications are image processing, signal processing, pattern recognition, medicine, military systems, financial systems, planning, control and search, and power systems. In Section 3.2.3, the advantages of ANN are listed and explained one by one. These benefits are non-linearity, input-output mapping, adaptivity, evidential response, contextual information, fault tolerance, VLSI implementation, uniformity of analysis and design, and neurobiological analogy. All of these advantages are the outcome of the ANN three natural features: massive parallel distributed computing architecture, knowledge storing ability in the weights, and learning capability to new environment.

In Section 3.3, the integration of artificial neural network to the Byzantine Agreement Protocol is discussed. The analogy of FCN as a graph is compared to the feed forward ANN with multi-layer perceptron. The formation of distributed ANN from the embedded ANN at each node of the distributed system is also delivered. In a nutshell, this chapter introduces the possibility and advantages of using the artificial neural networks to achieve the Byzantine Agreement.

CHAPTER 4: ARTIFICIAL NEURAL NETWORK BASED BYZANTINE AGREEMENT PROTOCOL

4.1 INTRODUCTION

The first Byzantine Agreement Protocol (BAP) to solve the consensus problem of Byzantine Generals Problem (BGP) was presented by Lamport, Shostak and Pease in 1982. Then variants of BAP are recommended to cope with different degrees of fault classes, different types of faults, and different kinds of network topologies. Wang and Kao (2001) proposed a new approach to Byzantine Agreement (BA) by using the neural network approach. Adopting the artificial neural networks (ANN) into the BGP has shown better performance over the traditional BAP. It shows great improvement in the requirements for space and time, parallel distributed computing capability at each node, learning and adaptability capability to dynamic Byzantine environment.

The application of ANN into the processor is possible due to the advanced development of VLSI and ULSI. A distributed system is a collection of computers or microprocessors via an interconnected network. The supercomputing power of the current microprocessors allows the imaginary ANN to be implemented in the hardware within promising time. When a processor in a distributed network has the ANN trained, the trained weights and biases of the neural network can be simply copied and stored into the processor, and makes the processor to act as an agent or problem solver for the consensus problem of BGP. All of the agents in the network integrate themselves among each other to form a multi-agent system (MAS) (O'Hare & Jennings, 1996).

In this thesis, the Lamport, Shostak and Pease's BAP (1982) is called traditional BAP. Meanwhile the Wang and Kao's BAP is called ANN based BAP. In this chapter, proposed ANN based BAP will be discussed in depth. Those subtopics of ANN based BAP, such as ANN architecture design, ANN learning algorithm, stages of BAP algorithm, experimental implementation, complexity, advantages and further improvement, will be brought into discussions one after the other.

4.2 DESIGN OF ARTIFICIAL NEURAL NETWORK ARCHITECTURE

A neural network is characterised by three main factors: neural architecture, activation function and learning algorithm. *Neural architecture* is the network topology or pattern of neuron connections. *Activation function* is the transfer characteristics of every single neuron. *Learning algorithm* is the method to train the weights on the connection links. In this section, both of the neural architecture and neuron activation function will be discussed together with the selections of ANN based BAP. The learning algorithm and its selection will be discussed in Section 4.3 then.

The neural network connectivity determines the neural architecture of ANN. There exist four kinds of neural architectures, which receive major concern: single-layer feed forward network, multi-layer feed forward network, recurrent network, and lattice structure.

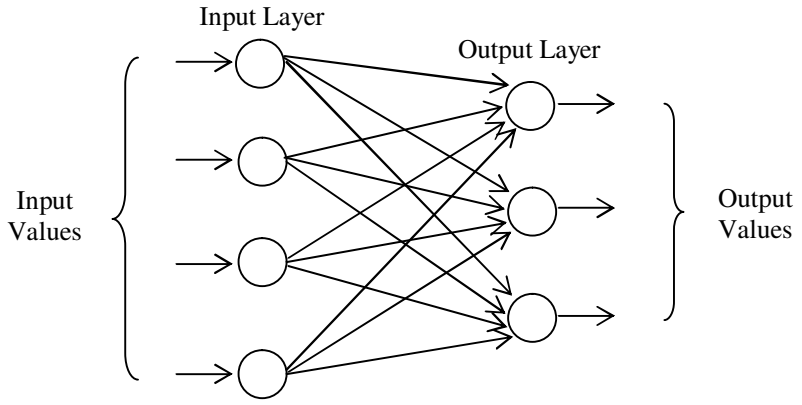


Figure 4.1 Fully connected single-layer feed forward network

Single-layer feed forward network is the simplest form of ANN. There is only one input layer of source nodes projecting onto an output layer of neurons in a single direction. This single direction gives the name of feed forward to this type of network. Normally the input layer is not counted in the total number of layers of neural networks. This is because no computation is carried out at the input layer. Since there is only one layer of output neurons and no hidden layer, there is only a

total of one layer of network. Figure 4.1 shows the architecture of single-layer feed forward network.

The second type of neural architecture is *multi-layer feed forward network*. It differs from single-layer feed forward network by having one or more hidden layers of hidden neurons. *Hidden neurons* intervene between the external input and the network output to enable the ANN for extracting higher order statistics. Therefore, the hidden neurons enable ANN to acquire a global perspective despite the local connectivity by having extra set of synaptic connections and extra dimension of neural interactions (Churchland & Sejnowski, 1992). This ability is very useful to those ANN having a large input layer or to simulate non-linear functions.

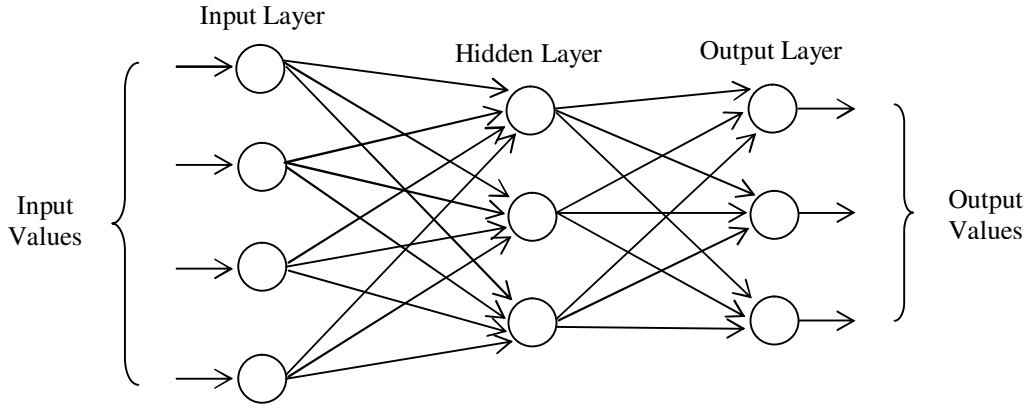


Figure 4.2 Fully connected two-layer feed forward network with one hidden layer

The input neurons in the input layer supply the input values to the hidden neurons in the first hidden layer. The output values of the first hidden layer will become the input values of the next layer. This next layer is the output layer for two-layer feed forward network. For ANN with layers more than two, the next layer will be the second hidden layer. The output values from the second hidden layer will then become the input values to the next layer until the output layer of the neural network. Generally, the neurons in each layer of the network get the input values from the sum of output values of the preceding layer only. The set of output values from the output neurons in the output layer constitute the overall ANN response to the first set of

input values in the input layer (set of original parameters from the external source). Figure 4.2 shows a two-layer feed forward network with one layer of hidden neurons.

For simplicity, the neural architecture in Figure 4.2 can be put into the notation of 4-3-3, where it indicates 4 input neurons, one hidden layer with 3 hidden neurons, and 3 output neurons. For a network of x-h_1-h_2-y, it means x input neurons in input layer, h_1 hidden neurons in the first hidden layer, h_2 hidden neurons in the second hidden layer, and y output neurons in the output layer.

Both the neural architectures in Figures 4.1 and 4.2 are *fully connected* structures. This is because each neuron in each network layer is connected to every other neuron in the adjacent forward layer. Figure 4.3 shows a partially connected two-layer feed forward network. It is *partially connected* since some of the weights or links are missing from the fully connected structure. The partially connected network is useful for locally connected network. Each hidden neuron is connected to a local set of input neurons lying within the immediate neighbourhood. The set of localized neurons, which feeds a neuron, constitutes the *receptive field* of the neuron. Similarly, each output neuron is connected to a local set of hidden neurons.

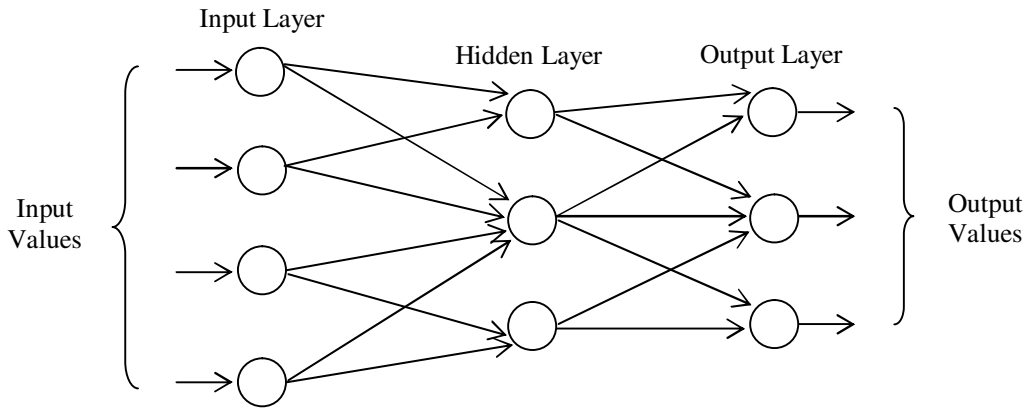


Figure 4.3 Partially connected two-layer feed forward network with one hidden layer

In the proposed ANN based BAP, the neural architecture being adopted is a two-layer feed forward network with one hidden layer. The single-layer feed forward network is insufficient in simulating the BGP. This is because the single-layer feed forward network is lacking of the capability to simulate complicated relationships

between the inputs and outputs. On the other hand, the multi-layer feed forward network (with 2 hidden layers or more) is more than enough and the drawback is that it is time consuming due to larger network connections.

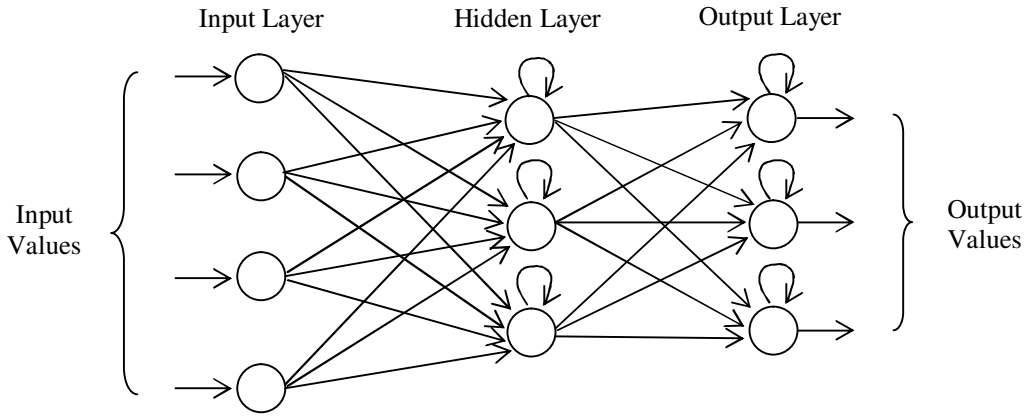


Figure 4.4 Recurrent network with self-feedback loop and 1 hidden layer

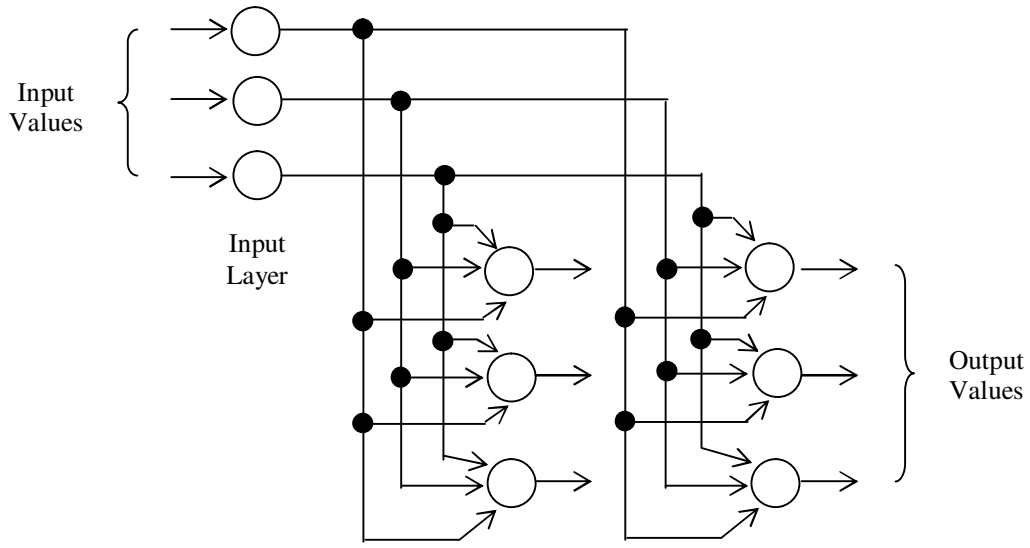


Figure 4.5 2-D lattice of 3-by-2 neurons

Since the other two structures are not in use, they are only discussed in brief. The third neural architecture is a *recurrent neural network*, which differs itself from feed forward network by having at least one *feedback* loop. The last structure is *lattice structure*. A lattice structure consists of 1-D, 2-D, or higher dimensional array of neurons with a corresponding set of input nodes supplying the input values to the array. The lattice dimension refers to the number of dimensions of the space where the graph lies. Figure 4.4 shows a recurrent network with self-feedback loop and one hidden layer. Figure 4.5 shows a 2-D lattice of 3-by-2 neurons.

For ANN, its computational complexity is called as *interconnection complexity*. Interconnection complexity is important to be considered when the ANN is scaled to handle larger problems. For a single-layer feed forward network with a structure of *x-y*, mapping *x* number of input parameters to *y* number of output parameters, its interconnection complexity will be *x*y* number of interconnections for a fully connected network. For small *x* and/or small *y*, the interconnection complexity is insignificant. However, as the scaling of a problem getting bigger, it is crucial to check the value of *x*y*. This is to ensure that the ANN training is possible to reach an optimal solution.

After the discussion on neural architecture, it is important also to select the correct activation function for each neuron in the ANN. *Activation function* is in fact the basic operation of a neuron that involves the sum of its weighted input values to produce an output value. Every neuron has a non-linear activation function except the input neurons in the input layer. Input neurons are special that their activation function is an identity function. Generally, the other neurons use the same non-linear activation function to achieve the advantages of multi-layer network. If linear function is used for the neurons in a multi-layer neural network, it is the same with the results of feeding input values through the single-layer neural network.

The types of activation functions are identity function, binary step function, binary sigmoid function, and bipolar sigmoid function. The linear function of identity function is shown in Equation 4.1 and Figure 4.6. Those non-linear functions are shown in Equations 4.2, 4.3, 4.4 and Figures 4.7, 4.8, 4.9 respectively.

Identity function: $f(x) = x$ for all x. (4.1)

Binary step function: $$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases} \quad (4.2)$$

where θ is the threshold

Binary *sigmoid function*: $$f(x) = \frac{1}{1+\exp(-\sigma x)} \quad (4.3)$$

where σ is the slope parameter.

Bipolar *sigmoid function*: $$f(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)} = \tanh(\sigma x/2) \quad (4.4)$$

where σ is the slope parameter.

Among all the activation functions, sigmoid functions (*S*-shaped curves) are the most useful functions. The logistic function and the hyperbolic tangent functions are the most common used in neural networks trained by back propagation neural network (BPNN). Its advantage is that the simple relationship between the value of the function at a point and the value of the derivative at that point reduces the computational burden during the training. For neural networks desiring output values of either binary or between the interval of 0 and 1, it is best to use a logistic function, which is a subset of sigmoid function ranging from 0 to 1. This logistic sigmoid function is given the name *binary sigmoid function* with a slope parameter, σ, which adjusts the steepness of the function.

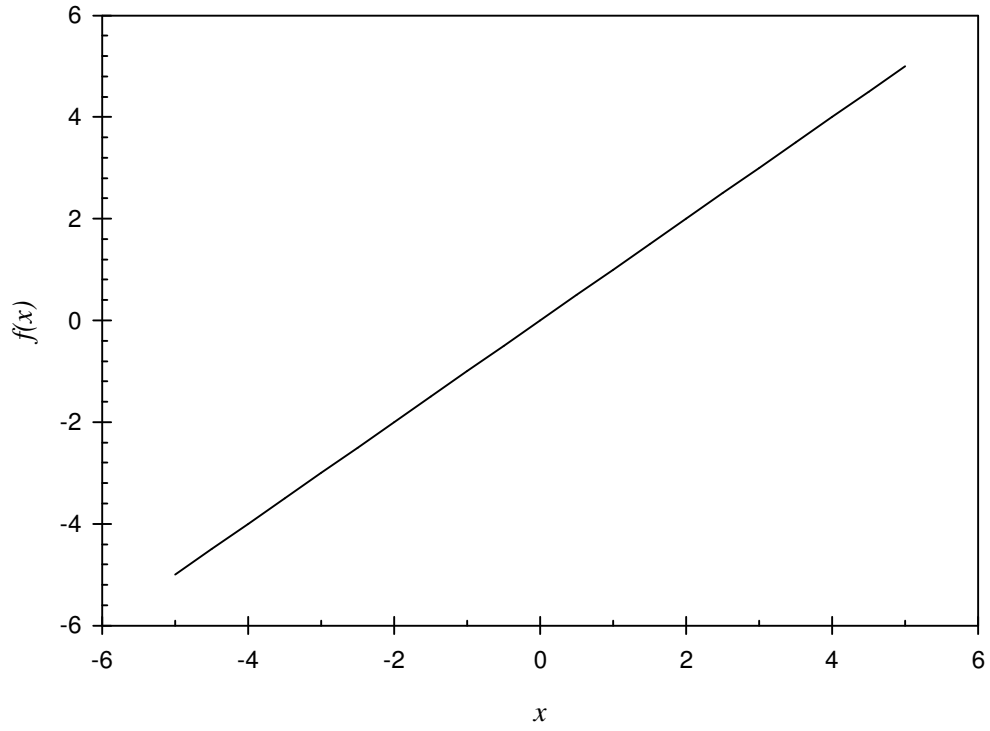


Figure 4.6 Identity function, $f(x) = x$

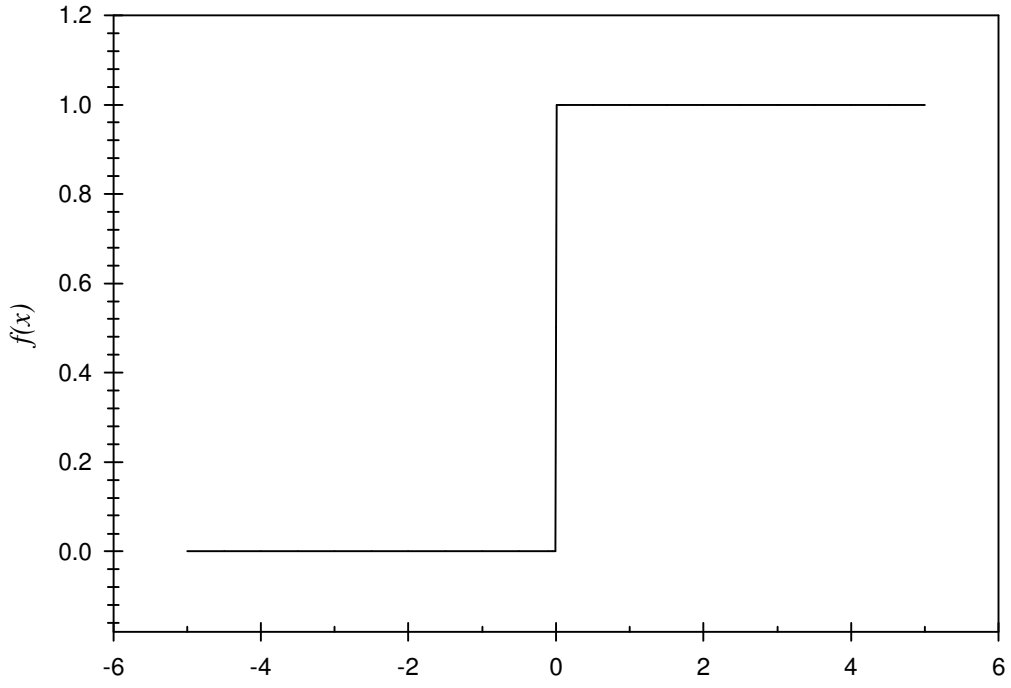


Figure 4.7 Step function, $f(x) = \begin{cases} 1 \text{ for } x \geq 0 \\ 0 \text{ for } x < 0 \end{cases}$

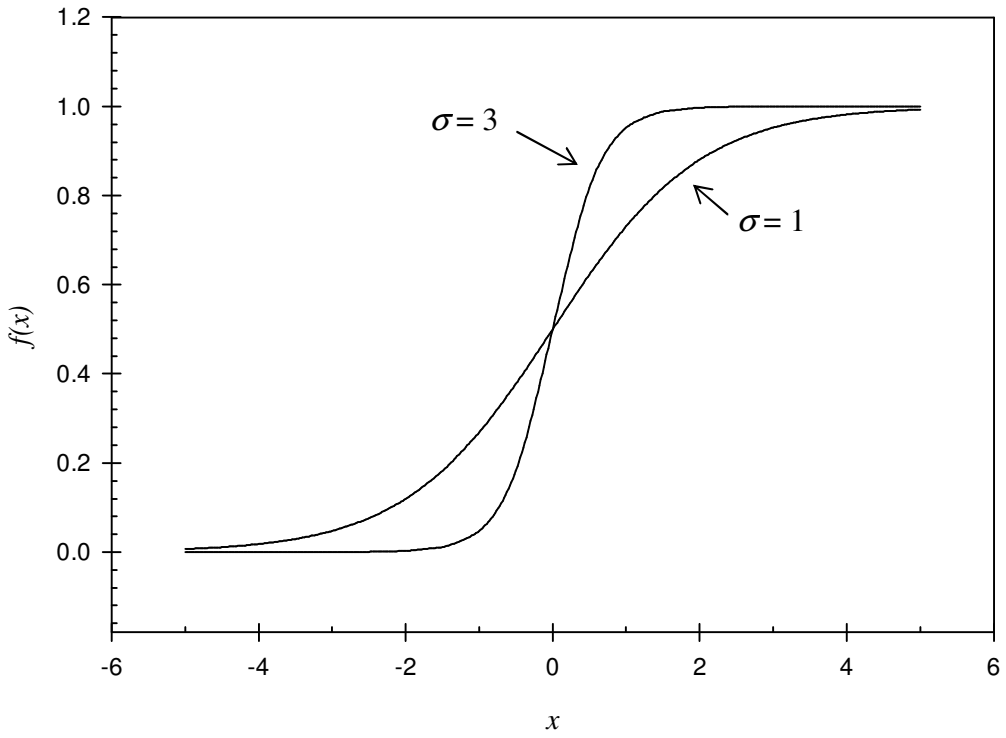


Figure 4.8 Binary sigmoid function, $f(x) = 1 / (1+\exp(-\sigma x))$ with $\sigma = 1, 3$

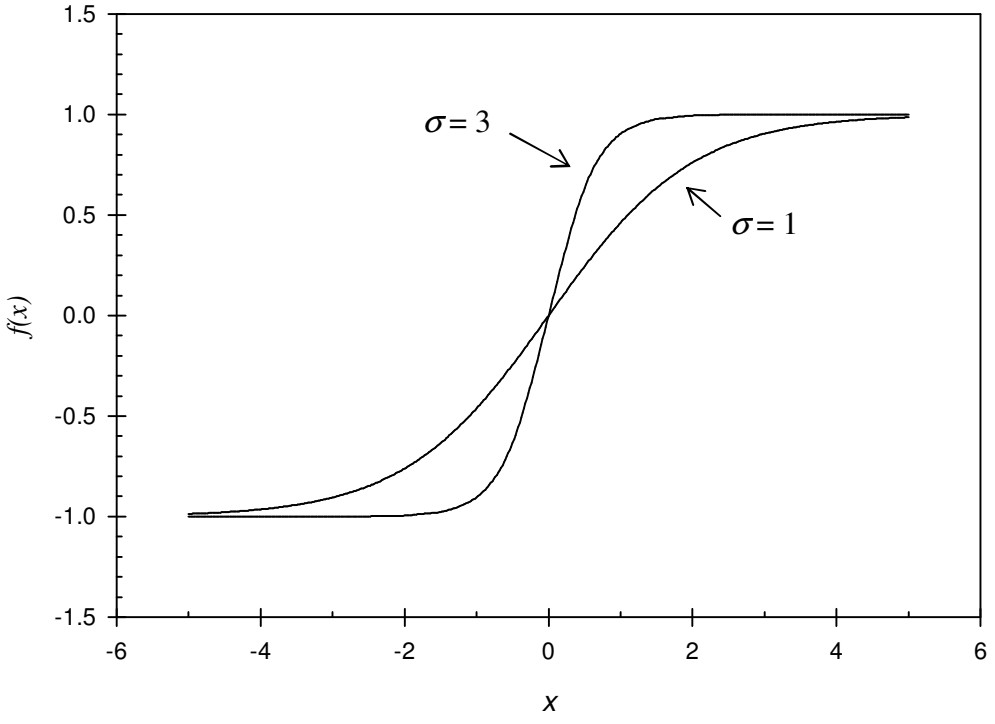


Figure 4.9 Bipolar sigmoid function, $f(x) = (1-\exp(-\sigma x)) / (1+\exp(-\sigma x))$ with $\sigma = 1, 3$

For binary sigmoid function in Equation (4.3), its derivative can be expressed in term of the function itself as in Equation (4.5). For bipolar sigmoid function in Equation (4.4), its derivative can be expressed in term of the function itself as in Equation (4.6). However, bipolar sigmoid function is best to be used for output values ranging from -1 to +1. Therefore, for ANN based BAP in this thesis, which aims for a Byzantine Agreement of binary bit "0" or bit "1", the binary sigmoid function is selected instead of the bipolar sigmoid function, though both of the activation functions have less computational burden because of easier representation of derivative.

Derivative of binary sigmoid function:

$$f'(x) = \sigma * f(x) * [1-f(x)] \tag{4.5}$$

Derivative of bipolar sigmoid function:

$$f'(x) = \sigma/2 * [1+f(x)] * [1-f(x)] \tag{4.6}$$

4.3 ANN TRAINING USING BACK PROPAGATION NEURAL NETWORKS (BPNN)

In Section 4.2, the neural architecture being opted for ANN based BAP is a two-layer feed forward network. The identity function is used for the activation function for the input neuron. For the other neurons, i.e. hidden neuron and output neuron, the selected activation function is binary sigmoid function. In this section, we will discuss on the weight initialization and learning algorithm.

Weight initialization is an important distinguishing characteristic of different ANN. Some types of the neural network have fixed weights without an iterative process. Some weights are changed as the time proceeds. The way to initialize the weights depends on the type of learning algorithm being chosen. Learning algorithm is also called as training method, and there are two categories of learning algorithm: supervised learning and unsupervised learning.

Supervised learning is the most typical ANN training method. The training is accomplished by presenting a set of training input values to match with a set of target output values. During the training, the set of training input values is repeatedly being applied to the network until the difference between the target output values and the real output values is within a pre-set error tolerance. When the error limit is reached, sets of input values, which are not yet being used, will be applied to the neural network for testing its generalization performance. Figure 4.10 shows the ANN supervised learning.

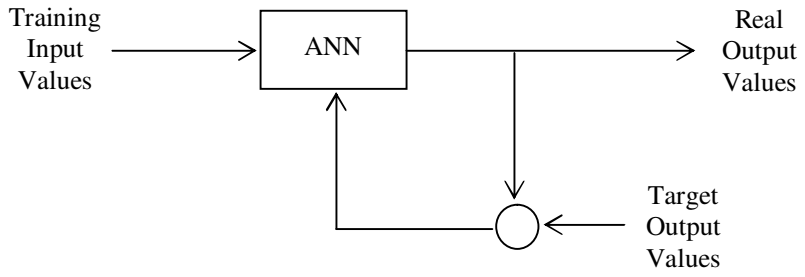


Figure 4.10 ANN supervised learning

For *unsupervised training*, there is no target output value as in supervised learning. A set of training input values is repeatedly applied to the network until a stable set of output values is obtained. Figure 4.11 shows the ANN unsupervised learning. This form of learning is very similar to the biological situation where there exists no target value normally.

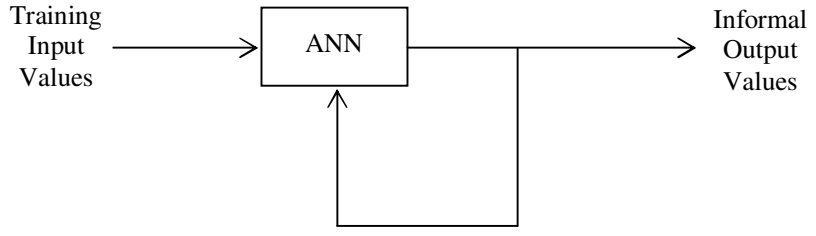


Figure 4.11 ANN unsupervised learning

In this thesis, the supervised learning algorithm is selected since the target output values are normally known. To train the ANN, we select the back propagation method, which is the most well known ANN learning algorithm. For a multi-layer feed forward network using the back propagation as the learning algorithm, it is called as a *back propagation neural network* (BPNN).

For function minimization, the gradient descent or steepest descent method is used. To train single-layer feed forward network, we use a gradient descent method called *delta rule*. The delta rule is modified to *generalized delta rule* in multi-layer feed forward network. In fact, BPNN uses a gradient descent technique too because there is generalized delta rule in the error minimization process. It is called back propagation because the error is backward propagated in refining the weights via the gradient descent method. The basic algorithm for BPNN is as shown below.

BPNN algorithm: *initialize weights and biases*
repeat
for each training pattern
train on that pattern
end for loop
until the error is acceptably low

The BPNN training algorithm is started from the initialization of weights and biases. Weights have been explained earlier as the links between neurons in a network for learning and storing the knowledge. What is bias and why we need it? The bias position in the neural network is shown in Figure 4.12. A *bias* is always set to one and connected to all the neurons except the input neurons.

As we know, every neuron in a layer has input values flowing from the previous layer except the input neurons. All of these input values from a previous layer have to be summed, passed through the activation function of the neuron in the present layer, before being delivered as an output value to the next layer. The activation function has a role to squash the summed input values into an allowable range. For binary sigmoid function, its range is from 0 to 1. This is the reason why

activation function is also known as *threshold function* or *squashing function*. The squashing function helps to squash those input values close to the extremes of the input range into a small part of the output range.

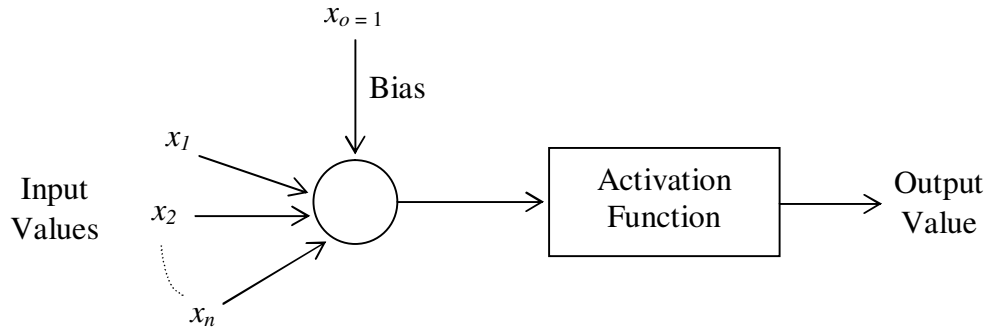


Figure 4.12 Bias of a neuron in the hidden layer or output layer

The role of the bias comes into the picture here. The bias job is to pull the summed input values to the hidden neurons and output neurons into the correct working range of the squashing function. The weights on the biases are modified the same way as the weights on the links between the neurons. However, bias is an optional component to the ANN. It helps to increase the speed of convergence but not a necessary part.

For the weight and bias initialization, there are random initialization and Nguyen-Widrow initialization. For *random initialization*, the choice of initial weights and biases will influence whether the ANN is trained to reach a global or local minimum of error, and how fast is the convergence. The update of the weight between two neurons relies on both the derivatives of the upper neuron and lower neuron activation function. Therefore, it is important in the option of initial weights so that the activation functions and the derivatives are not zero.

Furthermore, the initial weights cannot be too large to avoid the initial input values to the hidden neurons and output neurons falling into saturation region where the derivative of the sigmoid function has a very low value. On the contrary, the initial weights cannot be too small as well since it will cause an extremely slow learning rate. The common practice in random initialization is to have the weights

and biases being randomly initialized at the ranges of -0.5 to +0.5, -1 to +1, or some other set of values. It is a range having both the positive and negative values since the final weights can be either sign after the training.

Nguyen and Widrow (1990) demonstrated that a two-layer sigmoid or linear network can be viewed as an ANN that performs a piecewise linear approximation of any learned function. It is shown that weights and biases generated with certain constraint result in an initial network with better ability to form an approximation of an arbitrary function. Compared with purely random initialization, the Nguyen-Widrow initialization often shortens training time by more than an order of magnitude.

To have faster learning rate, the Nguyen-Widrow initialization is used in this research project. *Nguyen-Widrow* initialization is a modification from the random initialization based on a geometrical analysis of the response of the hidden neurons to a single input value. The geometrical analysis is then extended to the case of several input values by using Fourier transforms. Weights from the hidden neurons to the output neurons, as well as the biases on the output neurons, are commonly initialized to random values between -0.5 and +0.5. The weight initialization from the input neurons to the hidden neurons is designed to improve the learning ability of the hidden neurons. The initial weights and biases are distributed so that for each input pattern, the network input to one of the hidden neurons will be in the range in which the hidden neuron will learn most readily. The procedure of Nguyen-Widrow initialization is explained below. The settings of $\beta = 0.7$ and w_{ij}(old) = [-0.5,+0.5] are based on the geometrical analysis of the hidden neuron responses to a single input.

Let: p = number of input neurons X_i where $i = 1, 2, 3, \ldots, p$

q = number of hidden neurons Y_j where $j = 1, 2, 3, \ldots, q$

r = number of output neurons Z_k where $k = 1, 2, 3, \ldots, r$

β = scale factor where $\beta = 0.7(q)^{1/p}$

For each hidden neuron Y_j:

Initialize the weights w_{ij} from input neuron X_i to hidden neuron Y_j:

w_{ij}(old) = random number from -0.5 to +0.5

$\|w_j(\text{old})\| = [\ w_{1j}(\text{old})^2 + w_{2j}(\text{old})^2 + \ldots + w_{ij}(\text{old})^2 + \ldots + w_{nj}(\text{old})^2\ ]^{1/2}$

Re-initialize weights w_{ij} between input layer and hidden layer:

$$w_{ij}(\text{new}) = \beta * w_{ij}(\text{old}) / \|w_j(\text{old})\| \quad (4.7)$$

Set bias b_j on hidden neurons Y_j for $j = 1, 2, .., q$:

$$b_j = \text{random number between } -\beta \text{ and } +\beta \quad (4.8)$$

For each output neuron Z_k:

Initialize the weights w_{jk} from hidden neuron Y_j to output neuron Z_k:

$$w_{jk} = \text{random number between -0.5 and +0.5} \quad (4.9)$$

Set bias b_k on output neurons Z_k for $k = 1, 2, .., r$:

$$b_k = \text{random number between -0.5 and +0.5} \quad (4.10)$$

For the number of hidden neurons in the hidden layer, till now it is still an area of research. There is no rule in choosing the appropriate number for optimizing the ANN training except the trial-and-error approach. In this research, we let the number of hidden neurons, q, to be expressed in Equation (4.11).

$$q = \text{ceiling } [(p + r)/2] = \lceil (p+r)/2 \rceil \quad (4.11)$$

For the criterion on when to stop the ANN training, the mean squared error is used for error limit calculation. The *mean squared error* (MSE) is also called as training error or network error. MSE is in fact a function of the difference between the actual output values and the desired output values as stated in Equation (4.12) below.

$$\text{MSE} = \frac{1}{2}\sum_k (t_k - z_k)^2 \quad (4.12)$$

where t_k = target output value of neuron Z_k in the output layer

z_k = actual output value of neuron Z_k in the output layer

For BPNN algorithm implemented on a FCN two-layer feed forward network, it is divided into two halves. One is BPNN training algorithm and the other is BPNN application algorithm. Beginning with nomenclature, it is detailed as below.

Nomenclature

p-q-r Neural architecture with p units of input neurons X_i, q units of hidden neurons Y_j, and r units of output neurons Z_k.

w_{ij} Weight from the input neuron X_i in input layer to the hidden neuron Y_j in hidden layer.

b_j Bias to hidden neuron Y_j.

w_{jk} Weight from the hidden neuron Y_j in hidden layer to the output neuron Z_k in output layer.

b_k Bias to output neuron Z_k.

f(g) Activation function of the hidden neurons and output neurons with input g as in Equation (4.3).

$f'(g)$ Derivative of activation function *f(g)* as in Equation (4.5).

x_i Training input values where $x_i = x_1, x_2, x_3, \ldots, x_p$.

t_k Target output values where $t_k = t_1, t_2, t_3, \ldots, t_r$.

X_i Input neuron i with input value x_i and output value x_i as well.

Y_j Hidden neuron j with input value y_in_j and output value y_j.

$$y\_in_j = b_j + \sum_{i=1}^{p} w_{ij} x_i \tag{4.13}$$

$$y_j = f(y\_in_j) \tag{4.14}$$

Z_k Output neuron k with input value z_in_k and output value z_k.

$$z\_in_k = b_k + \sum_{j=1}^{q} w_{jk} y_j \tag{4.15}$$

$$z_k = f(z\_in_k) \tag{4.16}$$

MSE Mean squared error as in Equation (4.12).

α Learning rate

δ_k Portion of error correction to adjust weight w_{jk} and bias b_k due to an error at output neuron Z_k. It is also the information to be propagated back to the hidden neuron Y_j about the error at output neuron Z_k.

δ_j Portion of error correction to adjust weight w_{ij} due to the error information back propagated from the output layer to the hidden neuron Y_j.

BPNN Training Algorithm

Step 1: Normalize the input values.

Step 2: Initialize weights and biases using the Nguyen-Widrow initialization.

Step 3: While real MSE > pre-set MSE, do steps 4 to 11, and count the epoch.

Step 4: For each training pair, do steps 5 to 10

Feed forward:

Step 5: Each input neuron (X_i ; i = 1, 2, 3, …, p) receives input x_i and broadcasts it to all hidden neurons Y_j in the hidden layer.

Step 6: Each hidden neuron (Y_j ; j = 1, 2, 3, …, q) sums its weighted input values,

$$y\_in_j = b_j + \sum_{i=1}^{p} w_{ij} x_i \tag{4.17}$$

applies its activation function to get its output value y_j,

$$y_j = f(y\_in_j) \tag{4.18}$$

and sends output value y_j to all of the output neurons.

Step 7: Each output neuron (Z_k ; k = 1, 2, 3, …, r) sums its weighted input values,

$$z\_in_k = b_k + \sum_{j=1}^{q} w_{jk}\, y_j \tag{4.19}$$

applies its activation function to get its output value z_k.

$$z_k = f(z\_in_k) \tag{4.20}$$

Back propagation of error:

Step 8: Each output neuron (Z_k ; k = 1, 2, 3, …, r) receives a target pattern corresponding to the training input pattern, computes its error information term δ_k,

$$\delta_k = (t_k - z_k) * f'(z\_in_k) \tag{4.21}$$

calculates its weight correction term denoted by Δw_{jk},

$$\Delta w_{jk} = \alpha\, \delta_k\, y_j \tag{4.22}$$

calculates its bias correction term denoted by Δb_k.

$$\Delta b_k = \alpha\, \delta_k \tag{4.23}$$

Step 9: Each hidden neuron (Y_j ; j = 1, 2, 3, …, q) sums its delta inputs, δ_in_j, from the output neurons,

$$\delta\_in_j = \sum_{k=1}^{r} \delta_k\, w_{jk} \tag{4.24}$$

multiplies by the derivative of its activation function to calculate the error information term denoted by δ_j,

$$\delta_j = \delta\_in_j * f'(y\_in_j) \tag{4.25}$$

calculates its weight correction term denoted by Δw_{ij},

$$\Delta w_{ij} = \alpha\, \delta_j\, x_i \tag{4.26}$$

calculates its bias correction term denoted by Δb_j.

$$\Delta b_j = \alpha\, \delta_j \tag{4.27}$$

Update weights and biases:

Step 10: Each output neuron (Z_k ; k = 1, 2, 3, …, r) updates its bias and weights (j = 1, 2, 3, …, q).

$$b_k(\text{new}) = b_k(\text{old}) + \Delta b_k \tag{4.28}$$

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk} \tag{4.29}$$

Each hidden neuron (Y_j ; j = 1, 2, 3, …, q) updates its bias and weights (i = 1, 2, 3, …, p).

$$b_j(\text{new}) = b_j(\text{old}) + \Delta b_j \tag{4.30}$$

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \Delta w_{ij} \tag{4.31}$$

Step 11: Test the stopping criterion for the mean squared error MSE.

$$\text{MSE} = \frac{1}{2}\sum_{k=1}^{r}(t_k - z_k)^2 \tag{4.32}$$

Real MSE < pre-set MSE?

If YES, the BPNN training has finished. Go to step 12.

If NO, train the neural network again. Go to step 3.

Step 12: Store the neural architecture including the updated biases and weights: b_j(new), b_k(new), w_{ij}(new), and w_{jk}(new).

Recall the stored biases and weights from the BPNN training algorithm for task accomplishment via ANN implementation.

BPNN Application Algorithm

Step 1: Load the stored neural architecture including the biases and weights from the BPNN training algorithm.

Step 2: For each input value, do steps 3 to 5.

Step 3: For i = 1, 2, 3, …, p: set activation of the input neuron X_i.

Step 4: For j = 1, 2, 3, …, q:

$$y\_in_j = b_j + \sum_{i=1}^{p} w_{ij} x_i \tag{4.33}$$

$$y_j = f(y\_in_j) \tag{4.34}$$

Step 5: For k = 1, 2, 3, …, r:

$$z\_in_k = b_k + \sum_{j=1}^{q} w_{jk}\, y_j \tag{4.35}$$

$$z_k = f(z\_in_k) \tag{4.36}$$

Step 3: From the set of output values z_k, decide the information carried by set of input values x_i.

4.4 ANN TRAINING USING MODIFIED BPNN

The learning algorithm of BPNN discussed in Section 4.3 is inefficient in terms of computational time and epoch to train a neural network for satisfied accuracy and convergence speed. This situation can be improved by introducing some new terms to the BPNN. The BPNN with additional terms, such as momentum, adaptive learning rate and slope parameter, is called as *modified BPNN*. Modified BPNN exceeds standard BPNN in the speed of achieving the global minimum.

For back propagation method using *momentum*, μ, the weight change is in a direction combining the current gradient and the previous gradient. This modification allows faster training speed when some training inputs are very different from the majority of the inputs. Momentum is a value ranges from 0 to 1, where $\mu = [0,1[$. In the initial training stage, a large momentum shall be used to acquire a faster training speed. When it comes to the end of the training, it has to be reduced to avoid the oscillation problem. To use BPNN with momentum, more than one previous training set of weights is needed. The simplest form of BPNN with momentum for weight update at step (t+1) is shown below:

$$w_{jk}(t+1) = w_{jk}(t) + \alpha\delta_k y_j + \mu[w_{jk}(t) - w_{jk}(t-1)] \tag{4.37}$$

or $$\Delta w_{jk}(t+1) = \alpha\delta_k y_j + \mu\Delta w_{jk}(t) \tag{4.38}$$

and $$w_{ij}(t+1) = w_{ij}(t) + \alpha\delta_j x_i + \mu[w_{ij}(t) - w_{ij}(t-1)] \tag{4.39}$$

or $$\Delta w_{ij}(t+1) = \alpha\delta_j x_i + \mu\Delta w_{ij}(t) \tag{4.40}$$

As long as the corrections are in the same general direction for several patterns, momentum allows the network to make reasonably large weight adjustment. This helps to improve the case of using a small learning rate for preventing a large response to the error from any training pattern. Momentum also reduces the chance of stopping at a local minimum. This is because when the momentum is used, the neural network is trained to proceed not in the gradient direction, but in the combined direction of the current and previous gradients for weight correction. Nevertheless, there are some drawbacks of using momentum. The learning rate puts an upper limit on weight adjustment. The weight may be changed in an error increasing direction (Jacobs, 1988).

Adaptive learning rate, α, helps to speed up the BPNN convergence by changing the learning rate of the training (Cheung, Lustig & Kornhauser, 1990). There are a few criteria in the setting of adaptive learning rate. DeRouin, Brown, Beck, Fausett and Schneider (1991) recommended an approach to increase the learning rate whenever the training patterns from the under-represented classes are presented. In the standard approach of BPNN, duplication or creating noisy copies of the training patterns from the under-represented classes is not practical. However, for ANN based BAP, the BGP it deals with is by nature includes all the classes of possibility because of the arbitrary nature of Byzantine faults. Therefore DeRouin et al. (1991) approach cannot be used here. Another adaptive learning rate approach is the delta-bar-delta algorithm proposed by Jacobs (1988). This is an approach to have each weight trained on its own learning rate. Besides, the learning rates vary with time as training progresses. This method helps to decrease the epoch in a big scale. However, its drawback is that it is of high possibility that it may not converge during the training. Therefore, it is not used here as well. Instead, a simple adaptive learning rate approach is used based on the pre-set MSE. For different levels of MSE, different learning rate is assigned. At a high MSE, a high learning rate is used. As the epoch counted and the network being trained to approach the pre-set MSE closer and closer, the learning rate is reduced slowly from high to low in the range of 0 and 1.

In addition to momentum and adaptive learning rate, BPNN training can be improved by using the slope parameter. *Slope parameter*, σ, scales the range of the activation function so that it extends beyond the largest and smallest target values.

Refer to the activation function in ANN based BAP, i.e. binary sigmoid function in Equation (4.3), $\sigma = 1$ for standard BPNN, and $\sigma > 1$ to extend the limit of the activation function. By changing the σ, the slope of the function can be modified to cover any desired range centred at (x=0, $f(x)$=0.5). The derivative of modified binary sigmoid function is in Equation (4.5), and the example of $\sigma = 3$ is in Figure (4.3). In general, an optimal σ exists for fastest convergence. Small σ will have a slow convergence and too big the value of σ will cause the oscillation problem.

To optimize the parameters of modified BPNN, the only way is to do it empirically (Cooke & Lebby, 1998), since till now the parameter setting of modified BPNN is still an area to be explored. In general, the rule of thumb is to have large weight adjustment in the initial training period by having higher learning rate and momentum to avoid local minimum. Then, both parameters are gradually reduced according to the MSE error limit to get rid of the oscillation phenomenon. For slope parameter, it is determined empirically.

In a nutshell, the application of modified back propagation network (modified BPNN), using additional parameters, such as momentum, adaptive learning rate, and slope parameter, shows a better performance over the standard BPNN. For the ANN based BAP in this thesis specifically, the neural architecture of a fully connected two-layer feed forward network converges at a faster rate by adopting the learning algorithm of modified BPNN.

4.5 ANN BASED BAP

The traditional Byzantine Generals Problem (BGP) has been explained in depth throughout Chapter 2. In Chapter 3, the integration of artificial neural network (ANN) to Byzantine Agreement Protocol is brought forward. The discussion on ANN learning algorithm of BPNN is spread from Section 4.1 to 4.3. In this section, the innovative approach of artificial neural network based Byzantine Agreement Protocol (ANN based BAP) will be discussed in an algorithm divided into five stages.

These five stages are initialization stage, message exchange stage (or communication stage), training stage, application stage, and compromise stage. The initialization stage, message exchange stage and compromise stage are in fact more or less the same procedures in traditional BAP. Meanwhile, training stage and application stage are the ANN stages using BPNN algorithm. These two ANN stages are embedded between message exchange stage and compromise stage to express the integration of ANN to BAP to solve the consensus problem in the existence of arbitrary faults (BGP). The relationships of these stages are interpreted clearly via the block diagram in Figure 4.13.

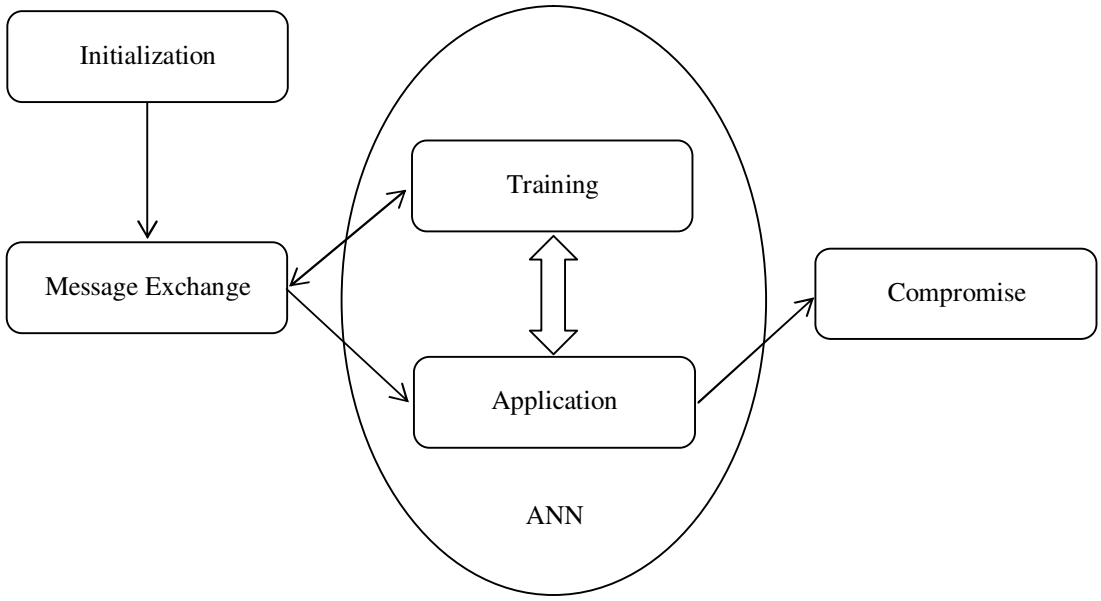


Figure 4.13 Block diagram of the ANN based BAP

It is to note that for BGP with single binary bit "0" and "1" as the messages, the numerical values involved in the rounds of message exchange to achieve the Byzantine Agreement are of binary bits as well. Therefore, the input values and the output values of the neural network is also of either binary bit "0" or "1". Subsequently, the gradient descent based BPNN is hence a special neural network handling input values to the input layer and output values from the output layer in binary field (Lursinsap & Kim, 1991).

The simplest distributed system for BGP study is a fully connected network (FCN) with four processor nodes. If n denotes number of nodes in a network, then n = 4 for the smallest FCN. Figure 4.14 shows the FCN model of a 4-processor network. C denotes the commander node. L_1, L_2 and L_3 denote the lieutenant nodes. For FCN-4 model, the corresponding neural architecture to be integrated to the ANN based BAP is illustrated in Figure 4.15.

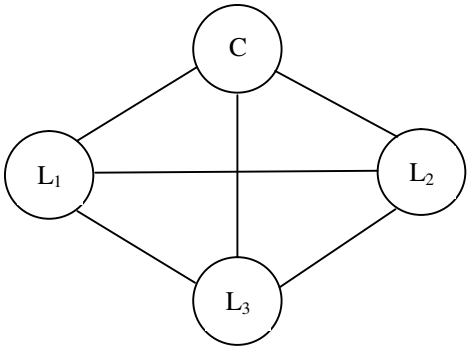


Figure 4.14 FCN model of a 4-processor distributed network

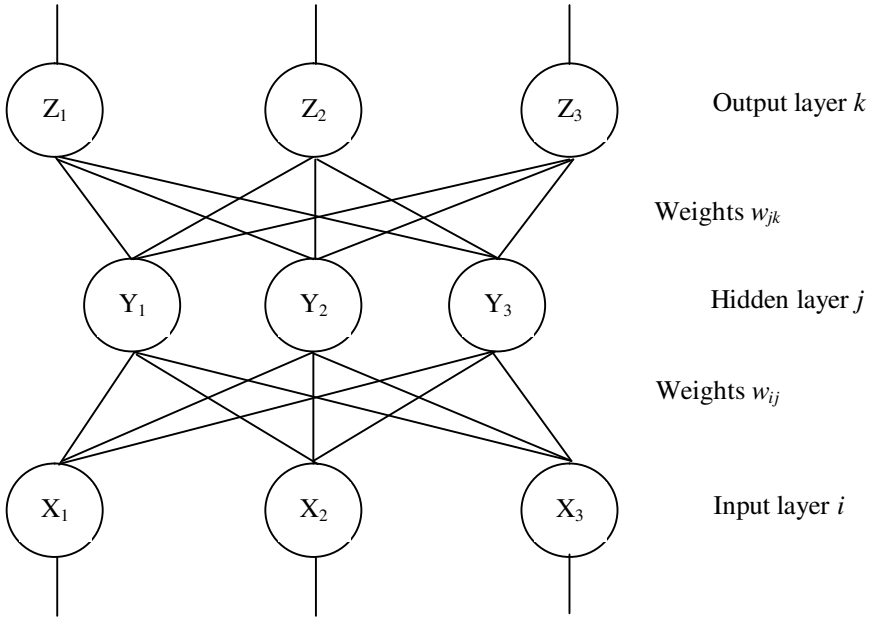


Figure 4.15 ANN model of a 4-processor distributed network

It is a fully connected two-layer feed forward network with neural architecture 3-3-3. The weights between input layer i and hidden layer j are w_{ij}. The weights between hidden layer j and output layer k are w_{jk}. On the other hand, the three input neurons (X_i : X_1, X_2 and X_3) in the input layer i represent the three lieutenant nodes (L_1, L_2 and L_3). The number of the hidden neurons (Y_j : Y_1, Y_2 and Y_3) in the hidden layer j follows from Equation (4.11). The three output neurons (Z_k : Z_1, Z_2 and Z_3) in the output layer k represents the Byzantine Agreement (BA) of bit "0", bit "1" and DEFAULT.

To show how ANN based BAP works, the simplest distributed system is adopted. It is a FCN-4 model in Figure 4.14 with L_3 as the faulty node. Commander node C, lieutenant nodes L_1 and L_2 are assumed to be loyal nodes. From Section 4.5.1 to 4.5.5, the algorithm of ANN based BAP is discussed stage by stage.

4.5.1 Initialization Stage

In the initialization stage, the number of processors in a distributed system is firstly determined. Then a corresponding ANN is created in every processor. For an n-processor system, the neural architecture will be (n-1) input nodes in the input layer, three output neurons in the output layer, and $\lceil (n+2)/2 \rceil$ hidden neurons in the hidden layer. For FCN-4, its architecture is 3-3-3.

Then, let the Byzantine Agreement (BA) to be achieved among all the loyal nodes as a single binary bit "0" or bit "1". For cases where a node receives no value, and fails to have the majority bit value from a set of received value, a pre-set value called DEFAULT is used. DEFAULT can be either bit "0" or bit "1".

4.5.2 Message Exchange Stage

In term of rounds of message exchange or rounds of communication, the proposed ANN based BAP in this thesis is different from the Lamport, Shostak and Pease's traditional BAP (1982), and Wang and Kao's BAP (2001). Both Lamport et al. (1982) together with Wang and Kao (2001) interchange message in the form of single bit value throughout all the rounds of message exchange. Therefore, both BAP

require (m+1) rounds of message exchange at least to guarantee the achievement of Byzantine Agreement (BA), where m is the number of faulty nodes. Applying the techniques from Yan, Wang, Chin (1991) and Wang, Yan (2000), we can modify the form of message delivery so that it needs only three rounds of communication for the proposed ANN based BAP to achieve the BA.

For ANN based BAP, the first round and second round of message exchange is the same with the traditional BAP as well as Wang and Kao's BAP. However, at the third round or final round of the message exchange, each node in the ANN based BAP broadcasts a string of received bit values to all the other nodes.

For an n-processor distributed system, it can be simulated into a graph of n fully connected network (FCN-n). In the first round of message exchange, the commander node sends its bit message to all the other (n-1) lieutenant nodes. Each lieutenant node is now having one bit of message. Subsequently in the second round, each lieutenant node sends its one bit of message to the other (n-2) lieutenant nodes. Now we have all together (n-1) bits of messages at every lieutenant node. In the third round or final round, each lieutenant node delivers its (n-1) bits of messages to all the other (n-2) lieutenant nodes. At the end of message exchange stage, the total bits of messages, Msg_{ANN}, of all the (n-1) lieutenant nodes is given by Equation (4.41).

$$\begin{aligned} Msg_{ANN} &= (n\text{-}1)*[1+(n\text{-}2)+(n\text{-}1)(n\text{-}2)] \\ &= (n\text{-}1)^3 \end{aligned} \tag{4.41}$$

To make the procedure clearly understood, the example of a 4-processor distributed system used in Section 4.5.1 is further developed here. For the worst case of a malicious network with four nodes, a maximum of one faulty node can be tolerated according to Equation (2.3). Assuming that the commander node C is loyal, then one of the three lieutenant nodes (L_1, L_2 and L_3) can be a faulty node. Due to the symmetrical property of the FCN model and ANN model, these three faulty situations are equivalent to each other. For instance, let the lieutenant node L_3 be the faulty node as assumed in Section 4.5.1. L_3 is then able to send out arbitrary messages to confuse the other loyal lieutenant nodes.

In the first round of message exchange, the commander node *C* broadcasts its source messages bit "1" to the other lieutenant nodes L_1, L_2 and L_3. The flow of the messages is shown in Figure 4.16.

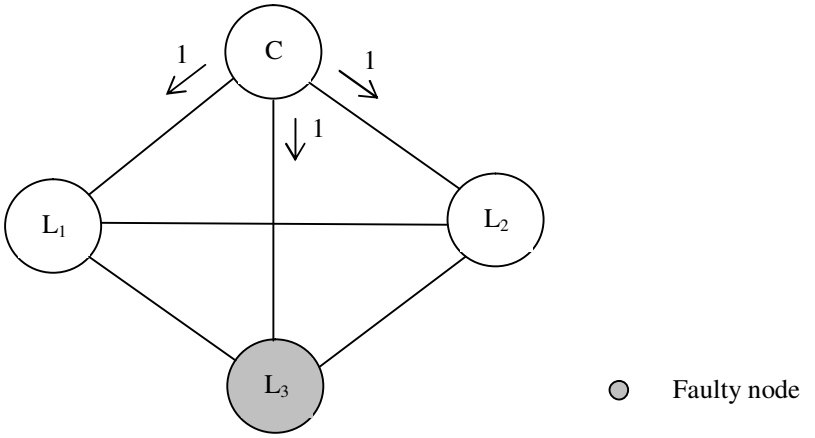


Figure 4.16 First round of message exchange for a 4-processor distributed network

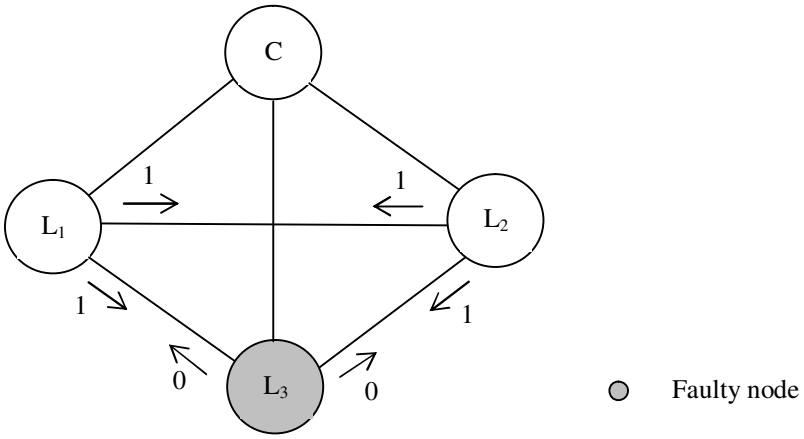


Figure 4.17 Second round of message exchange for a 4-processor distributed network

Then, in the second round of message exchange, the loyal lieutenant nodes L_1 and L_2 send the received value in the first round to the other lieutenant nodes to confuse the to-be-made common consensus. The message flow in the second round is shown in the Figure 4.17.

During the last round of the message exchange, each lieutenant node delivers a 3-bit string to the other lieutenant nodes. The process of this third round of message exchange is illustrated in Figure 4.18. With the combination of all the bit values received, a message exchange matrix (MEM) is completely formed within each processor or node at the end of the message exchange stage.

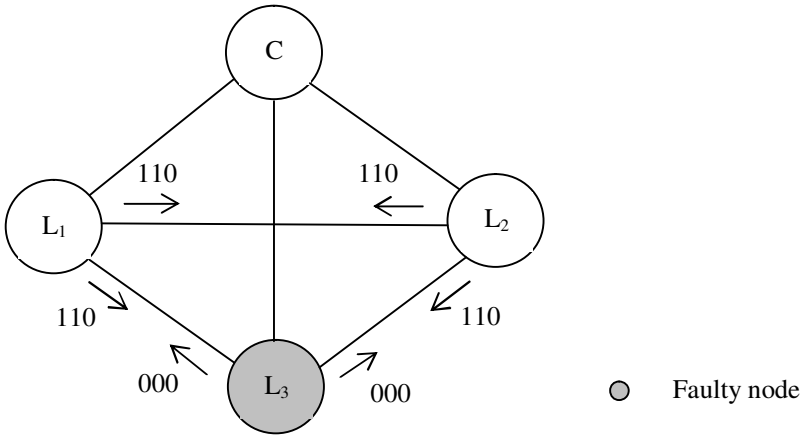


Figure 4.18 Third round of message exchange for a 4-processor distributed network

4.5.3 Training Stage

For a distributed network where the embedded neural network is not yet trained, the MEM formed in the message exchange stage will be used in the ANN training stage. MEM is a square matrix with the dimension of (n-1)x(n-1) for an n-node network. For the example of FCN-4, the generated MEM is a 3x3 matrix.

The MEM matrix is used for the set of training input values to the designed neural network. A set of MEM together with its corresponding target output values is used for the BPNN training algorithm in the ANN training stage. The row vectors of the MEM with the dimensions of 1x(n-1) are the training input vectors to be inserted into the input neurons in the input layer. These training input vectors from the row vectors of the MEM have their corresponding target output vectors. The target output vectors are supposed to be the output values from the output neurons in the output layer of the neural network.

Using the set of training vectors and target vectors, the embedded ANN is trained by using the gradient descent based BPNN as discussed in Section 4.3. The BPNN learning algorithm is repeated as long as the calculated mean squared error (MSE) is larger than the pre-set MSE error limit. For a MEM of FCN-n, there are always one MSE corresponding to every row of the MEM. Hence, there are a total of (n-1) MSEs at every node. Each MSE has to be less than the pre-set MSE before the learning process ends.

Row by row, each row vector of the MEM is fed to the input layer i of the BPNN. Each input neuron X_i processes the message it receives according to its identity activation function. Then, input neurons send the output values x_i of the input layer i to each hidden neurons Y_j in the hidden layer j, via the weights w_{ij}. Subsequently, the hidden neurons Y_j process the information and send the output values y_j of the hidden layer j to each output neuron Z_k in the output layer k via the weights w_{jk}. Both the hidden neurons and output neurons act as threshold logic units with a binary sigmoid activation function. Binary sigmoid function as in Equation (4.8) is chosen due to its computational efficiency.

There are two major processes in the BPNN training stage. One is the *feed forward process* and the other is the *back propagation process*. During the feed forward process, MSE is generated from the function of difference between the real output z_k and the target output t_k as in Equation (4.32). If the calculated MSE at either one of the rows of the MEM is larger than the pre-set MSE, the BPNN learning algorithm will be repeated.

Before the repeat, the back propagation process is initiated to use the calculated MSE for updating the weights and biases, i.e. w_{ij}, w_{jk}, b_j and b_k. The weight and bias adjustment is carried out via the gradient descent method. Then the BPNN is repeated from the feed forward process. The same set of training input values and target output values is used. However, the neural network now has an updated weights and biases. There will be another back propagation if the calculated MSE is still larger than the pre-set MSE. Both the feed forward and back propagation processes are repeated till the pre-set MSE is achieved. Now, the training stage ends, and the ANN is ready to solve the BGP in the ANN application stage.

4.5.4 Application Stage

In the training stage, the weights and biases after the pre-set MSE is achieved will be saved into every processor or node. The weights and biases will be recalled in the application stage to solve the arising BGP by reaching the interactive consistency conditions of (C.1) and (C.2) in Section 2.5.

The application stage is in fact the same with the feed forward process in the BPNN training stage. Once the trained weights and biases are loaded, each processor acts with an embedded ANN. For every session of source message delivery from the commander node, there will be three rounds of communication in the message exchange stage. The received messages through rounds of message exchange will generate a MEM at each node. The MEM will be used in the application stage together with the embedded ANN to find out the common consensus.

Each node will have the row vectors of the MEM to be fed into the trained neural network in the manner of row after another row. The output values z_k from the output neurons Z_k in the output layer k of each node will be delivered to the compromise stage. The set of output values z_k will help the loyal nodes to solve the BGP by gaining the Byzantine Agreement.

For instance, let us adopt the 4-processor distributed system in Section 4.5.1 and 4.5.2 to explain the description on training stage and application stage. For 4-processor system, it is a graph of FCN-4 with the maximum faulty node $m = 1$. From the implementation of the ANN based BAP, the MSE is recorded for each epoch. Different pre-set MSE against the required epochs in the BPNN training stage is plotted in Figure 4.19. The chosen pre-set epochs are 0.1, 0.01, 0.001, and 0.0001.

To see the epochs needed for BPNN training at each critical cases of $m = 1$, 2 and 3, the n-processor distributed systems with $n = 4$, 7 and 10 are chosen. For case $n = 4$, it is as discussed above. For the other two cases of $n = 7$ and 10, their graphs of MSE against epochs are plotted in Figure 4.19 as well. From Figure 4.19, it is noticed that the required epochs for BPNN training reduce as the number of nodes in a network increases from $n = 4$ to $n = 10$.

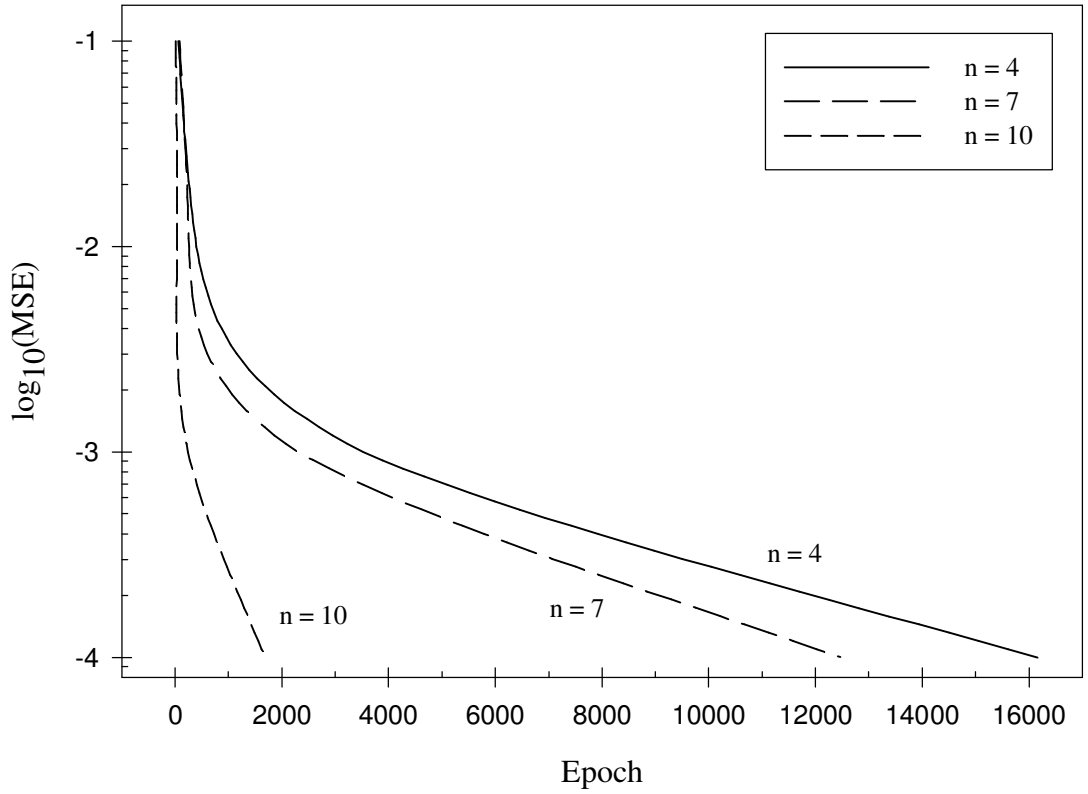


Figure 4.19 Required epochs for *n*-processor systems of $n = 4$, 7 and 10

4.5.5 Compromise Stage

In the compromise stage, the output values z_k from the output neurons Z_k in the output layer k will be used to determine the common agreement among all the loyal processors or nodes. Table 4.1 shows the executing results of a 4-processor system with the pre-set MSE of 0.0001. To have BGP solved, all the loyal nodes are required to achieve the common agreement. As a result, the consideration of the executing results of the loyal nodes is sufficient to ensure the success of ANN based BAP.

At lieutenant nodes L_1 and L_2, the MEMs formed are as below:

$$\text{MEM of } L_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.42}$$

$$\text{MEM of } L_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.43}$$

Row by row, the vectors of the MEM at node L_1 are fed into the trained neural network. Row [1 1 0], then [1 1 0], and at last [0 0 0] give three output values called the local majority values (local MAJ) of $[1\ 1\ 0]^t$ as in Table 4.1. At the same time, node L_2 carries out the same process on its MEM independently in parallel with node L_1.

Table 4.1 Executing results of a 4-processor system

Node	MEM	z_k			Local MAJ	Node MAJ	MAJ
		z_1	z_2	z_3			
L_1	1 1 0	0.004	0.995	0.003	1	1	1
	1 1 0	0.004	0.996	0.003	1		
	0 0 0	0.990	0.010	0.003	0		
L_2	1 1 0	0.004	0.995	0.002	1	1	
	1 1 0	0.004	0.996	0.003	1		
	0 0 0	0.999	0.001	0.003	0		

- Values of z_k (z_1, z_2, z_3) are rounded to the closest integer of "0" or "1".
- MEM : Message exchange matrix.
- MAJ : Majority.
- DEF : DEFAULT for the cases of no message received or no majority in a set of values received.

The value of local MAJ depends on the output values z_k from the output neurons in the output layer, i.e. z_1, z_2, and z_3. Since the theoretical output values z_k are in the format of binary bits "0" and "1". The experimental results of z_k as in Table 4.1 have to be rounded to the nearest integer of either "0" or "1". Then we can use the output layer interpretation as in Equations (4.44a-c) below to determine the local MAJ value.

If $z_1 z_2 z_3 = 100$, then local MAJ = bit "0". (4.44a)

If $z_1 z_2 z_3 = 010$, then local MAJ = bit "1". (4.44b)

If $z_1 z_2 z_3 = 001$, then local MAJ = DEFAULT value. (4.44c)

Generally, the number of local MAJ values at a node is (n-1). From the (n-1) local MAJ values, the bit value with the maximum frequency of occurrence is selected by applying the majority function. If there is no value or no majority bit in the set of local MAJ, then the DEFAULT value is assigned. The majority bit selected from the local MAJ set is called as node majority value (node MAJ). The node MAJ represents the final decision of a node on the message sent by the source node or commander node.

Refer to Figure 4.16, commander node C, lieutenant nodes L_1 and L_2 belong to the group of loyal node. Only lieutenant node L_3 is a faulty node. From Table 4.1, we notice that both the loyal lieutenant nodes L_1 and L_2 decide on message bit "1" as their final agreement. Therefore, the interactive consistency conditions (ICCs) of (C.1) in Section 2.5 is fulfilled since all the loyal lieutenant nodes L_1 and L_2 obey the same order "1". We check that loyal commander node C sends out source message "1" in the first round of message exchange. Hence, the interactive consistency conditions (ICCs) of (C.2) in Section 2.5 is fulfilled as well since the order of the loyal commander node C is obeyed by all the loyal lieutenant nodes L_1 and L_2.

As a result, the BGP of a 4-processor distributed system is solved since both the ICCs of (C.1) and (C.2) are satisfied. The Byzantine Agreement of the solved BGP is expressed by the majority value (MAJ) in Table 4.1, which is message bit "1" among all the loyal nodes.

4.6 IMPLEMENTATION OF ANN BASED BAP

The implementation of ANN based BAP on a 4-processor system is discussed throughout Section 4.5. Nevertheless, it is a discussion on how ANN based BAP runs instead of the efficiency of BPNN training algorithm. In this section, the BPNN training algorithm is tested under Pentium PC by using the Matlab programming language (Hagan & Demuth, 1995; MATLAB, 1997; Demuth & Beale, 1992-2000).

Standard BPNN and modified BPNN are compared for their performance by having extra parameters, such as bias (b), momentum (μ), adaptive learning rate (α) and slope parameter (σ). Both types of BPNN are compared in Table 4.2. These comparisons are carried out over the n-processor distributed systems for n = 4, 5, 6, …, 12 under the critical situation of maximum faulty nodes m to be tolerated. For m = 1, it is n = 4, 5 and 6. For m = 2, it is n = 7, 8 and 9. For m = 3, it is n = 10, 11 and 12. From these few examples, we can see the effect of increasing distributed network size over the required epochs for BPNN training algorithm of ANN based BAP.

Table 4.2 Differences between standard BPNN and modified BPNN

BPNN Parameters	Standard BPNN	Modified BPNN
Weight & bias initialization	Random	Random / Nguyen-Widrow
Bias, b	Optional	Compulsory
Momentum, μ	0.0	]0,1[or $0 < \mu < 1$
Adaptive learning rate, α	1.0	]0,1] or $0 < \alpha \leq 1$
Slope parameter, σ	1.0	≥ 1
Convergence speed	Slower	Faster

Before the experimental results, it is good to know about the architecture of the n-processor system and its corresponding neural network. The n-processor system is simulated into a graph of fully connected network (FCN-n), and the chosen

neural network is a two-layer feed forward network using gradient descent based BPNN. There are (n-1) input neurons, $\lceil (n+2)/2 \rceil$ hidden neurons, and three output neurons. FCN-n can tolerate a maximum of $m = \lfloor (n-1)/3 \rfloor$ faulty nodes. For case n = 4, its FCN and ANN models are shown in Figure 4.16 and 4.15 respectively. For comparisons, the cases of n = 5, 6, and 7 are used with the corresponding FCN and ANN models in Figures 4.20 and 4.21, Figures 4.22 and 4.23, Figures 4.24 and 4.25.

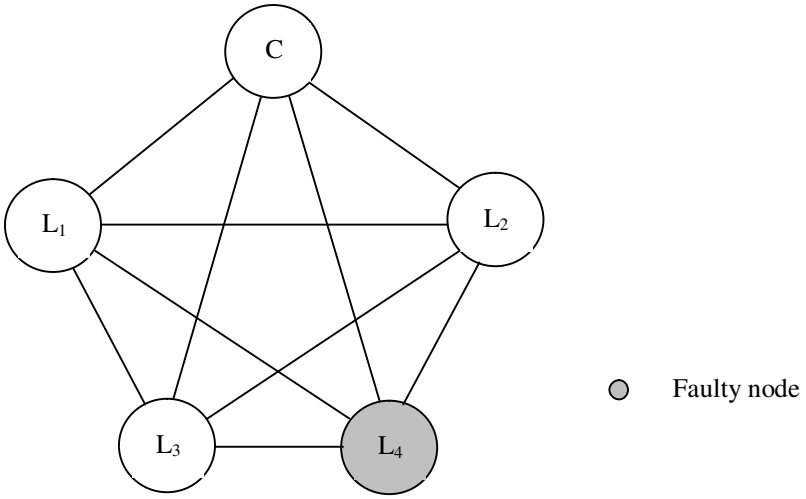


Figure 4.20 FCN model of a 5-processor distributed system

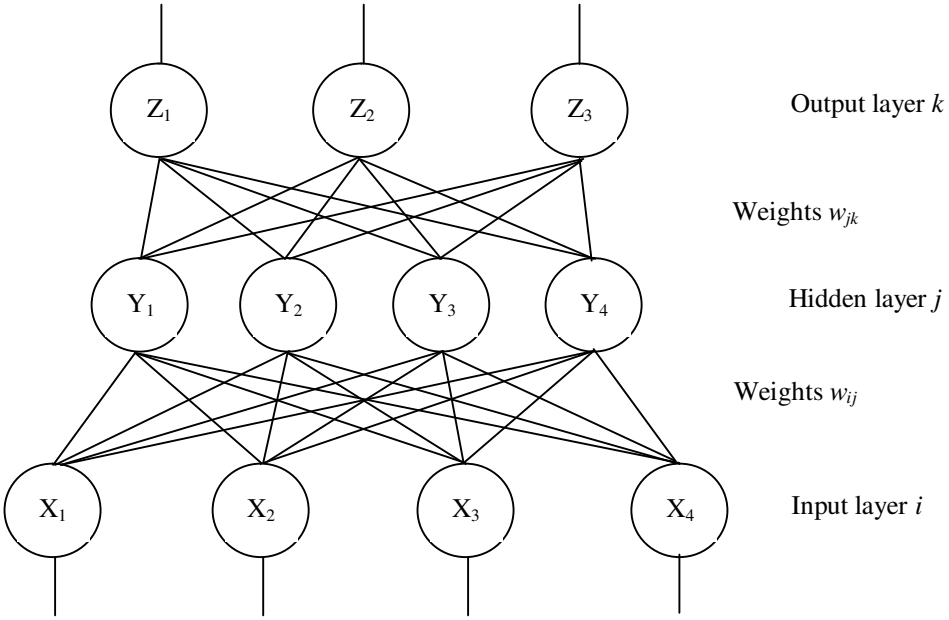


Figure 4.21 ANN model of a 5-processor distributed system

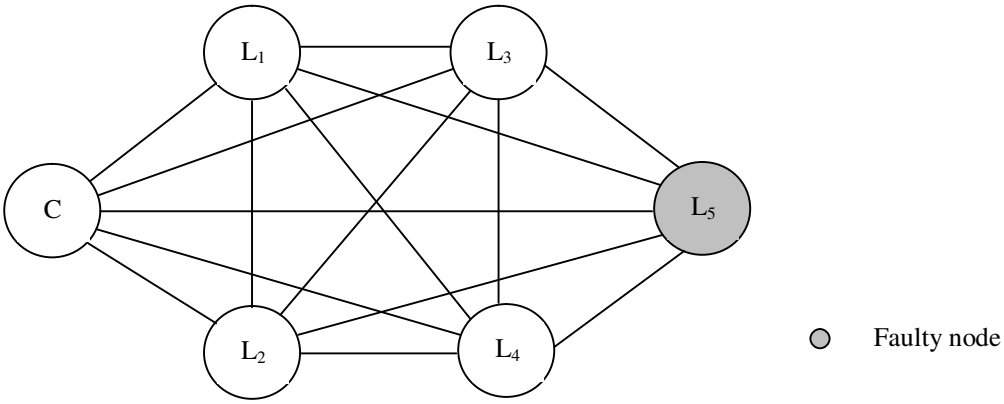


Figure 4.22 FCN model of a 6-processor distributed system

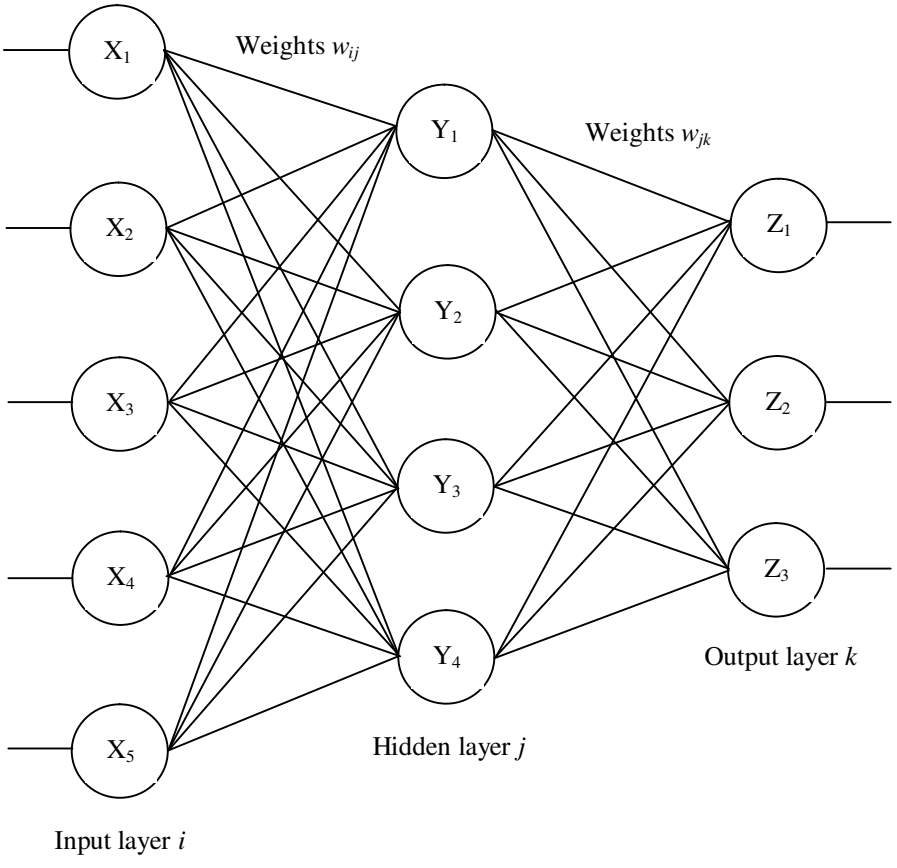


Figure 4.23 ANN model of a 6-processor distributed system

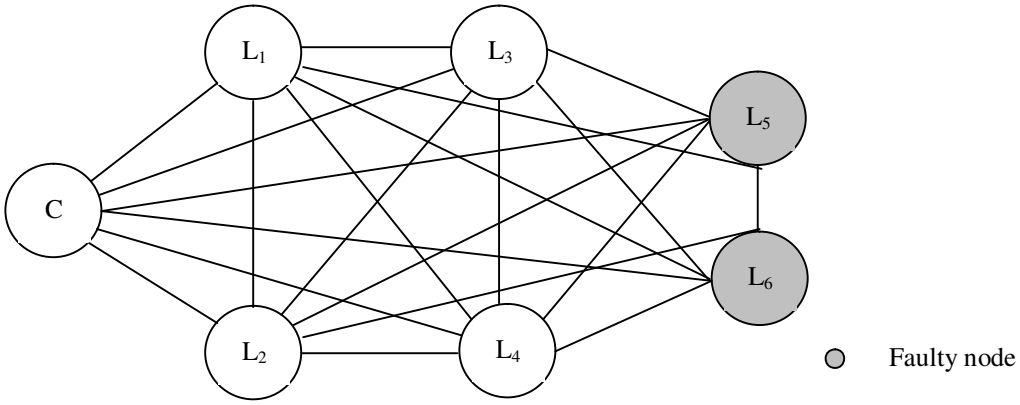


Figure 4.24 FCN model of a 7-processor distributed system

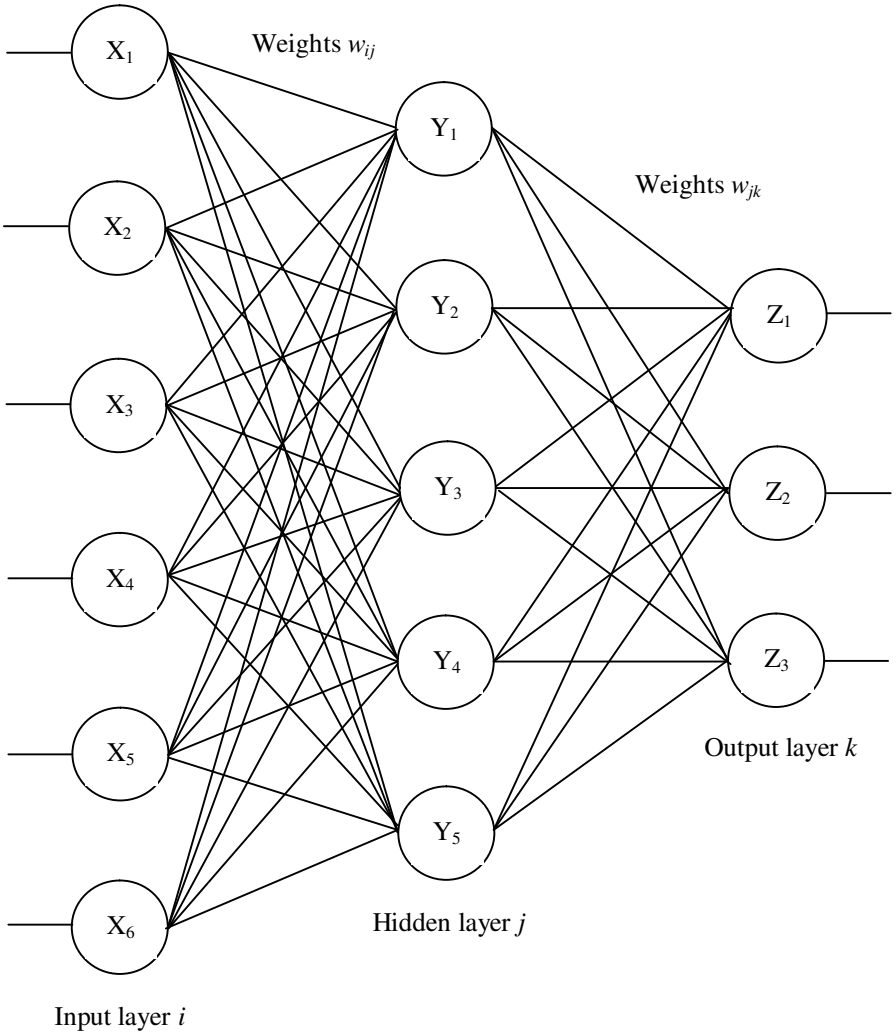


Figure 4.25 ANN model of a 7-processor distributed system

All the ANN based BAP experiments are run with the BPNN training algorithm stops at the MSE error limit of 0.001. To measure the number of epochs needed for training up a neural network more accurately, five tests are run for each set of experiment. The minimum, maximum and average epochs are then recorded as the experimental results. This is because the random initialization of weights and biases will affect the epoch needed for ANN training. The fluctuation of the epoch needed from the average epoch can be observed in Table 5.2.

However, the location of the faulty processor or the sequence number of a node to where it is connected to the input neuron in the input layer will not affect the epoch needed for ANN training. This is because a distributed system simulated into a graph of FCN model is naturally a symmetrical structure. On the other hand, the weights between any two layers of the BPNN map every node in one layer to every node in another layer. Hence, the ANN model is having the symmetrical property as well.

The symmetrical properties of FCN model and ANN structure make the location of each processor in the distributed system and the sequence number of each node to the input neurons of the embedded ANN to be trivial. Specifically these symmetrical properties apply to the faulty nodes as well. The information of knowing which node is faulty will not affect the required epoch for ANN training.

The crucial factor influencing the required epoch is the number of faulty nodes, *m*, within the distributed network. In a nutshell, the FCN-*n* models for FCN-4, FCN-5, FCN-6 and FCN-7 in Figure 4.16, 4.20, 4.22 and 4.24 are common models though the faulty node is assumed to be at a specific location. In the following section, the epoch needed for BPNN training of ANN based BAP will be presented in the graphical form.

4.6.1 Standard BPNN for No Bias and With Bias

In standard BPNN, bias is an optional component. However, the introduction of bias helps to ease the problem of having zero input values to the input neurons. This situation occurs most frequently in having the binary bits as the input. The zero input values may introduce the overshoot in the initial stage of training. The BPNN with bias has shown faster convergence in epoch as compared to BPNN without bias in Figures 4.26-4.32 as below. All the experimental results are based on standard BPNN training with parameters of momentum at 0.0, learning rate at 1.0, slope parameter at 1.0 and MSE error limit at 0.001. The figures shows faster response and the overcome of zero-input-value problem for BPNN training with bias.

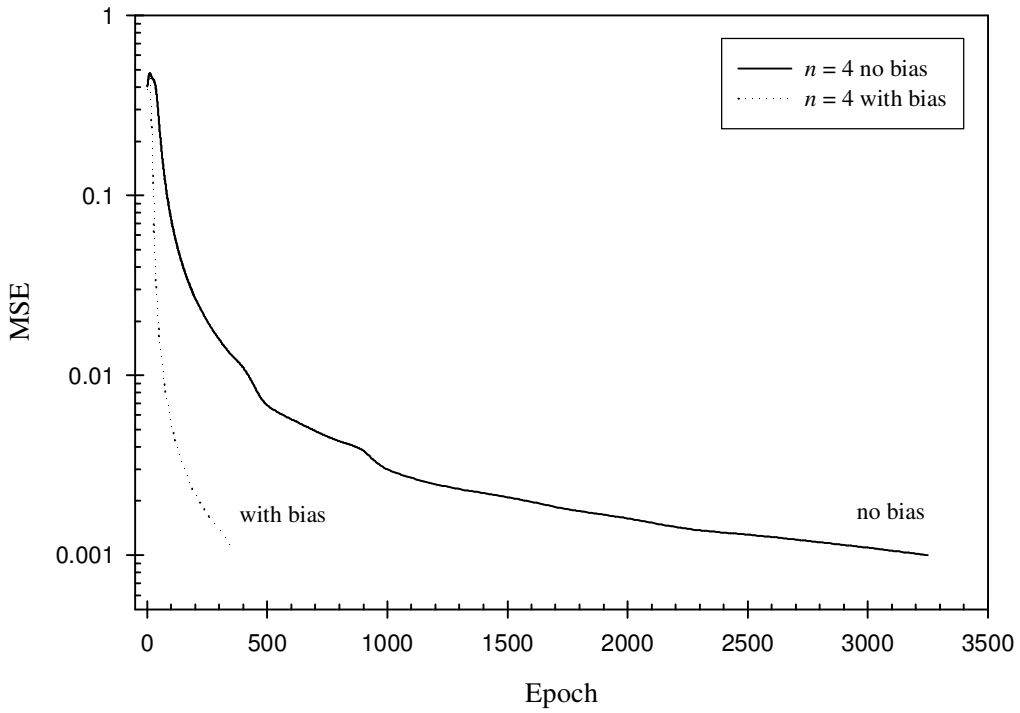


Figure 4.26 Required epoch for FCN-4 using BPNN without bias and with bias

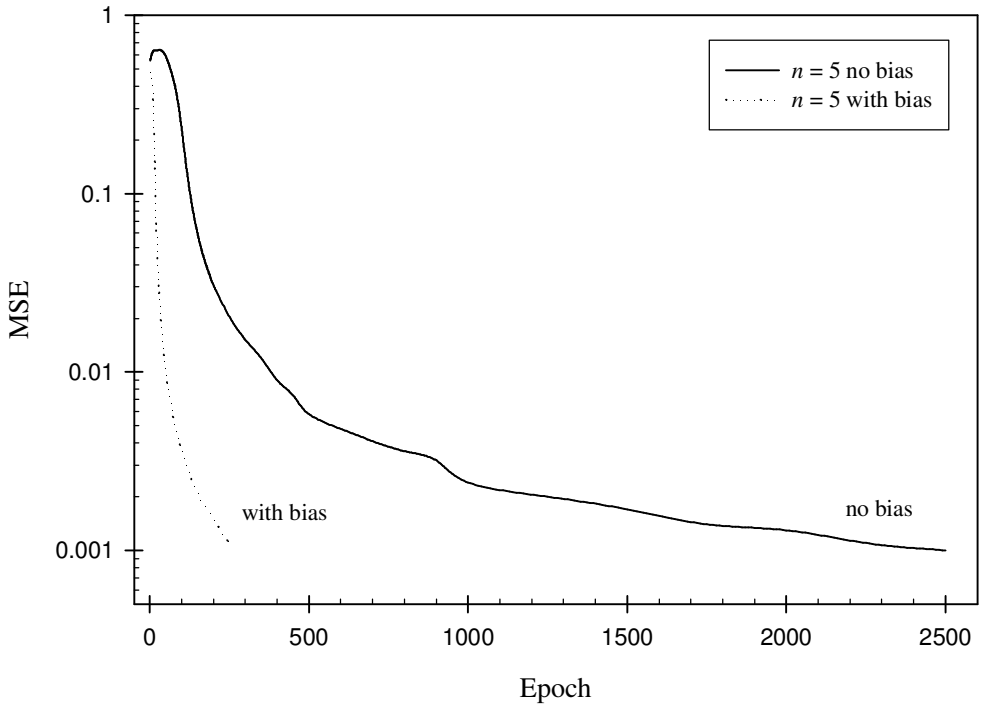


Figure 4.27 Required epoch for FCN-5 using BPNN without bias and with bias

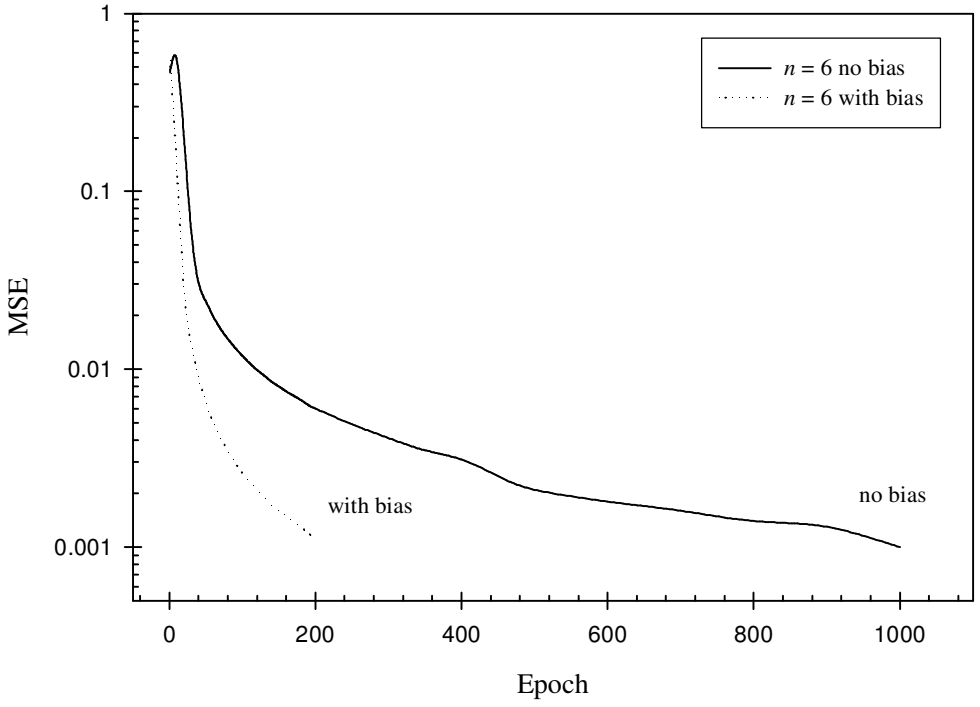


Figure 4.28 Required epoch for FCN-6 using BPNN without bias and with bias

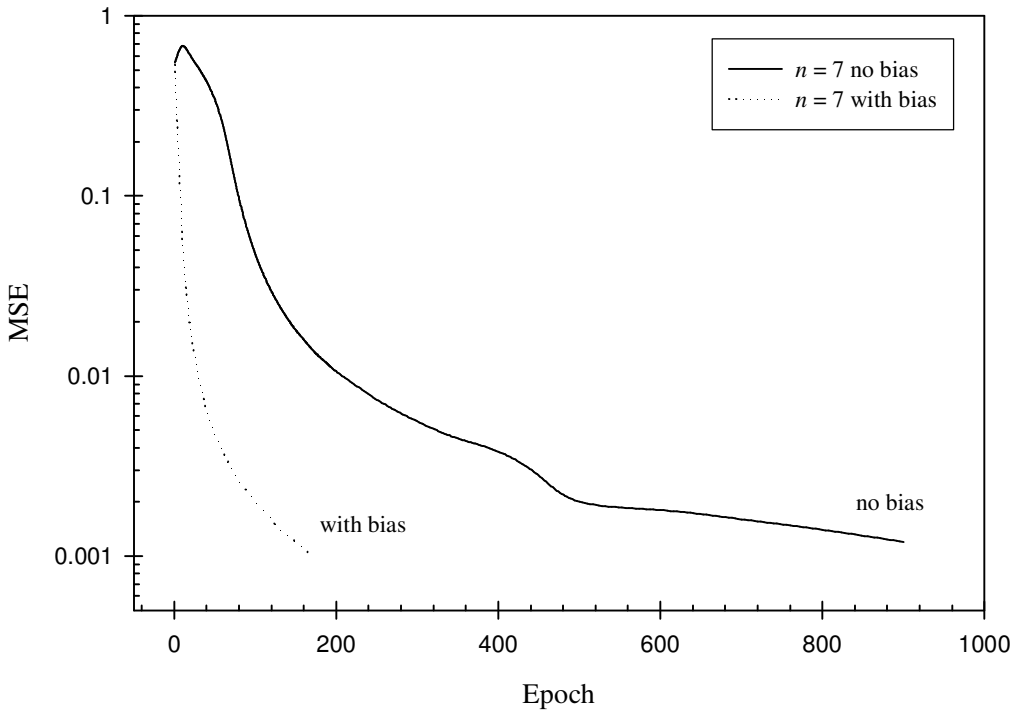


Figure 4.29 Required epoch for FCN-7 using BPNN without bias and with bias

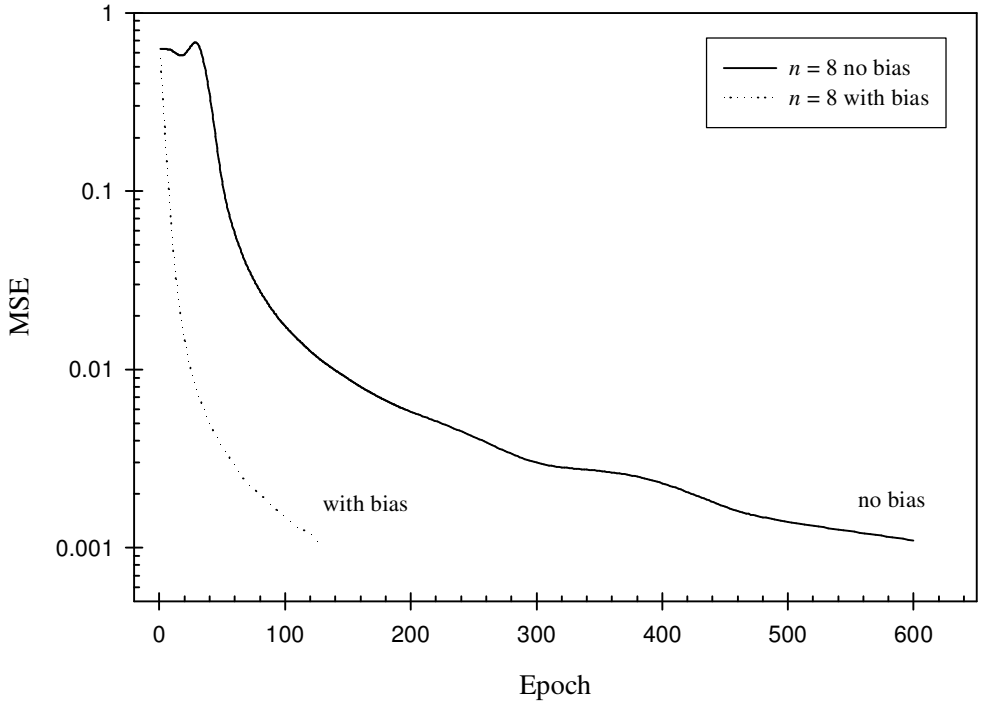


Figure 4.30 Required epoch for FCN-8 using BPNN without bias and with bias

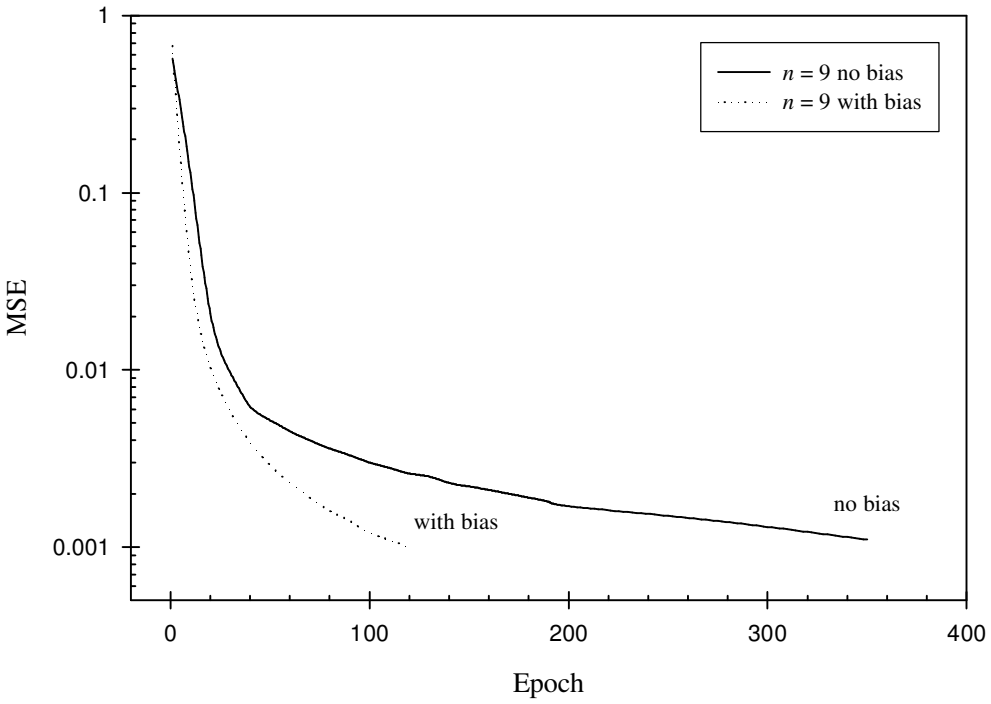


Figure 4.31 Required epoch for FCN-9 using BPNN without bias and with bias

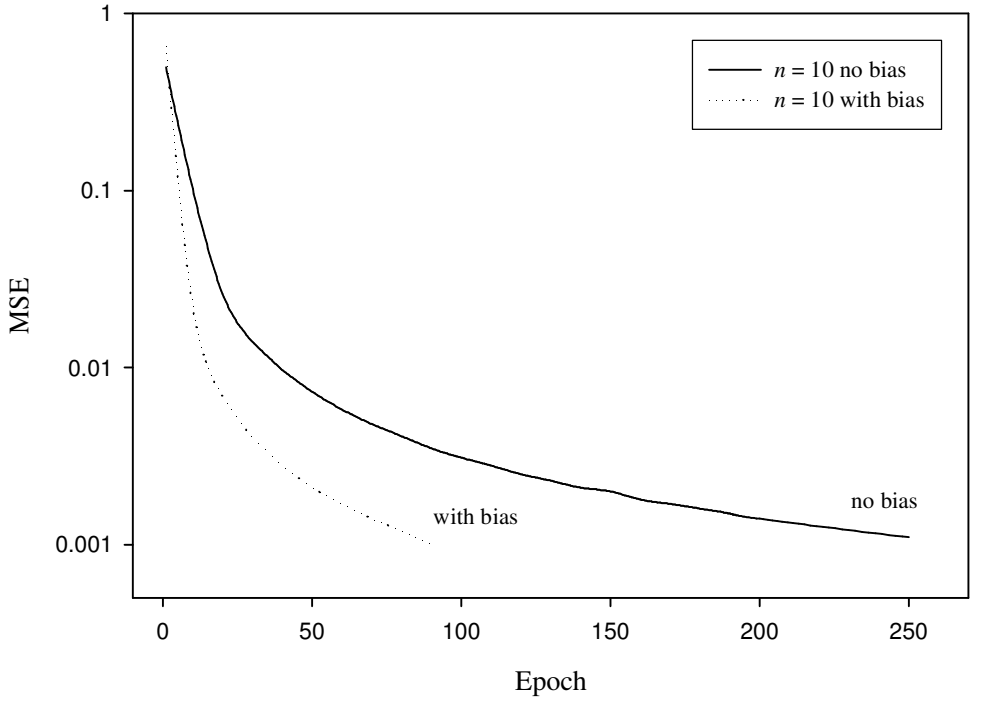


Figure 4.32 Required epoch for FCN-10 using BPNN without bias and with bias

4.6.2 BPNN with Random Initialization and Nguyen-Widrow Initialization

For standard BPNN training, the random initalization is normally used. To speed up the BPNN training of ANN based BAP, there is a special way of initialization recommended by Nguyen and Widrow (1990). This special method is only applicable to two-layer neural network having the sigmoid or linear activation function. It is called Nguyen-Widrow initialization and explained in Section 4.3. The performance of BPNN training with random initialization and Nguyen-Widrow initialization is compared in Figures 4.33-4.37 below. All the experimental results are based on standard BPNN training with parameters of momentum at 0.0, learning rate at 1.0, slope parameter at 1.0 and MSE error limit at 0.001.From the figures, the Nguyen-Widrow initialization converges faster than random initialization.

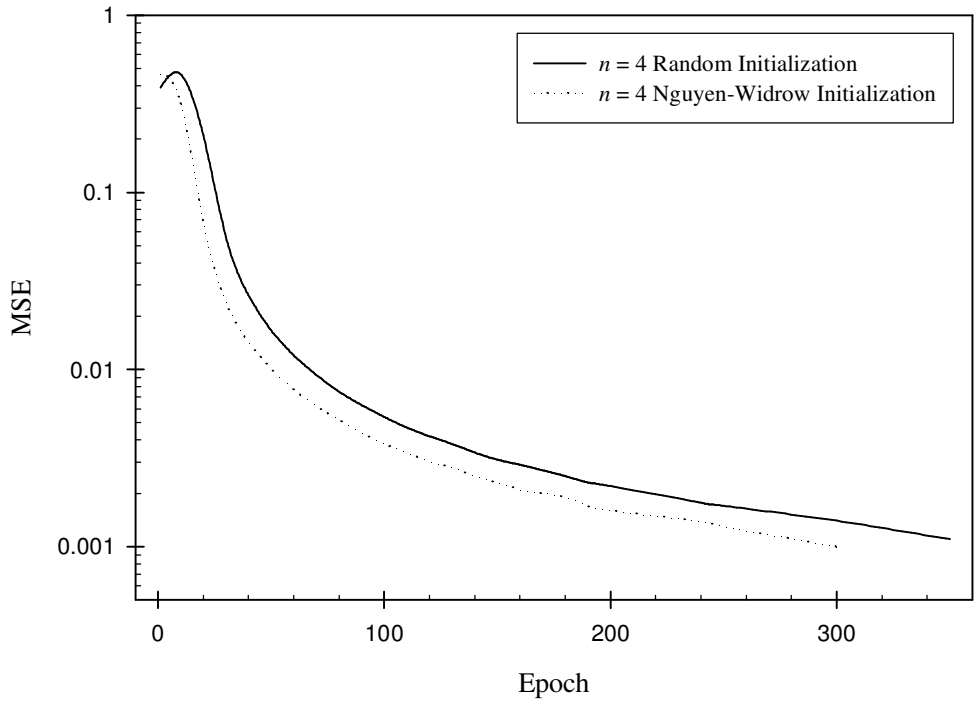


Figure 4.33 Required epoch for FCN-4 with random and Nguyen-Widrow initialization

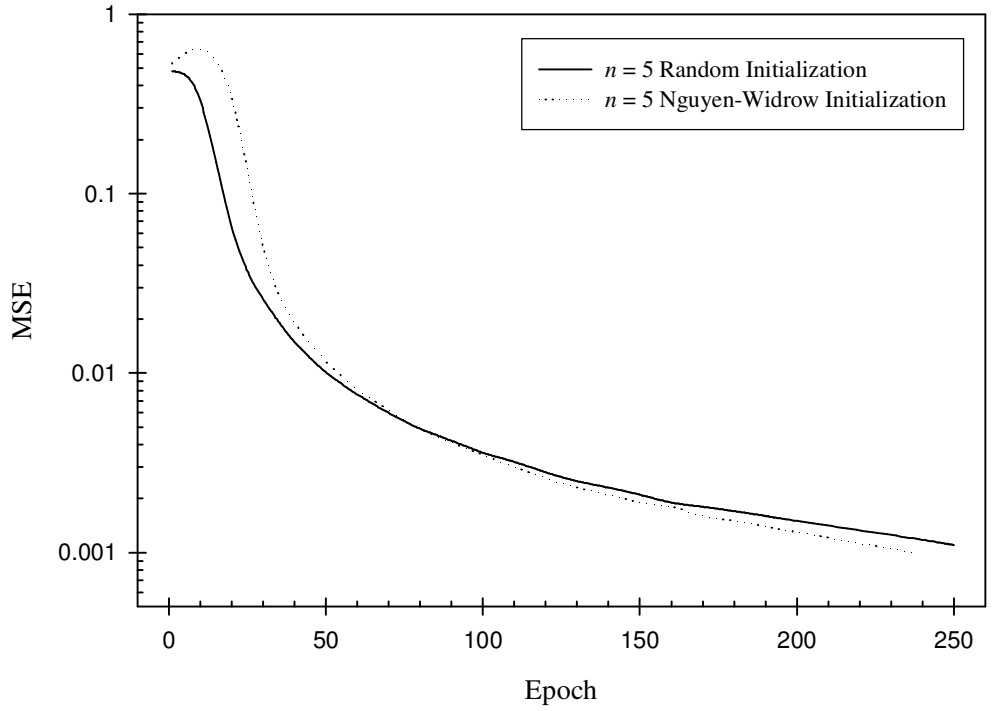


Figure 4.34 Required epoch for FCN-5 with random and Nguyen-Widrow initialization

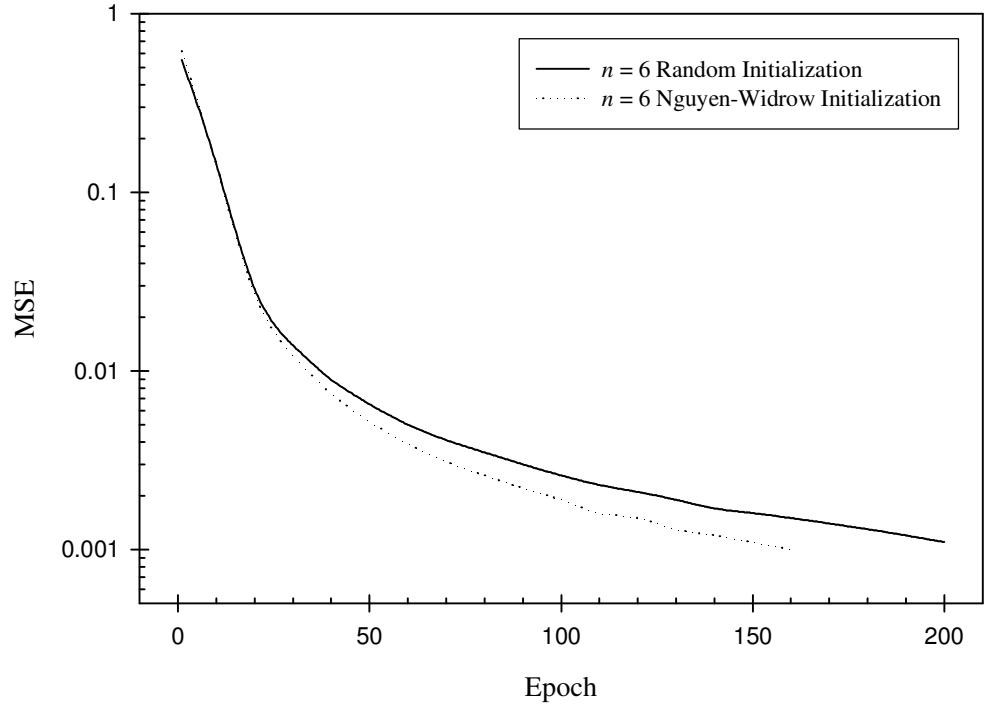


Figure 4.35 Required epoch for FCN-6 with random and Nguyen-Widrow initialization

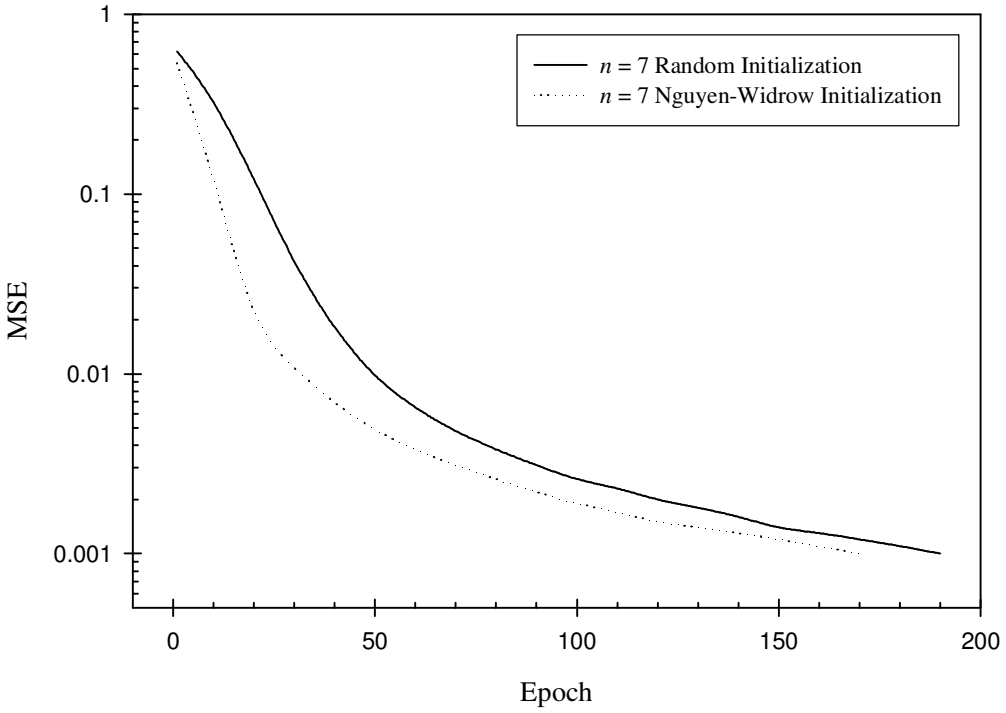


Figure 4.36 Required epoch for FCN-7 with random and Nguyen-Widrow initialization

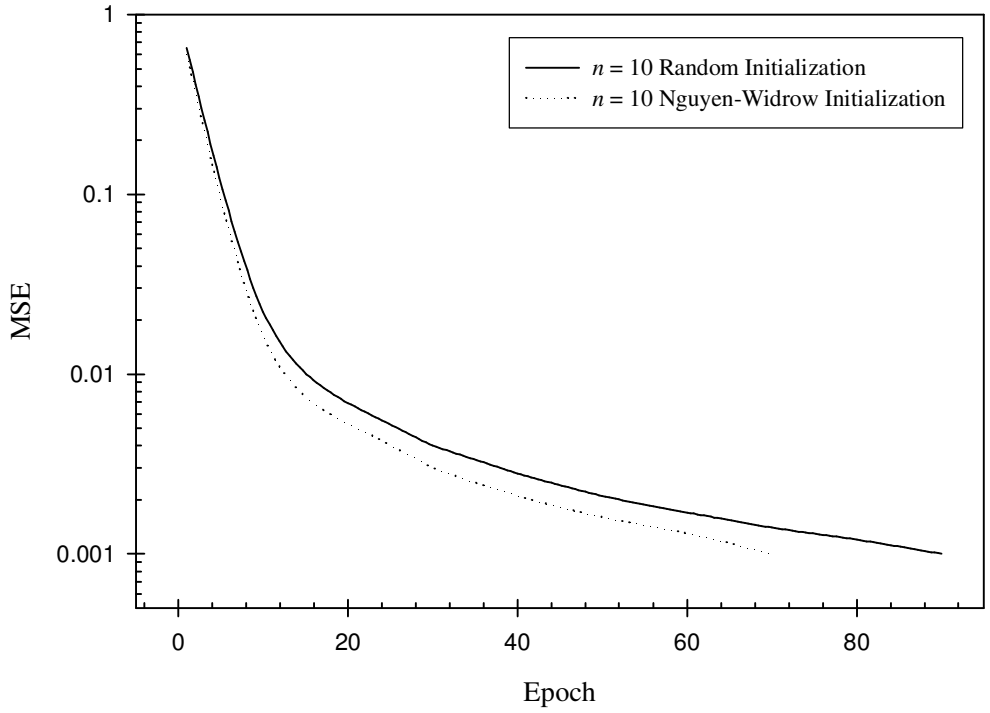


Figure 4.37 Required epoch for FCN-10 with random and Nguyen-Widrow initialization

4.6.3 Effects of Momentum to BPNN training

By introducing the parameter momentum, μ, into the standard BPNN training, we have a modified BPNN. Momentum helps to speed up the weight corrections for faster convergence by combining the current gradient and the previous gradient. For standard BPNN, $\mu = 0$. For modified BPNN, the momentum ranges from 0 to 1 excluding the values of 0 and 1, $\mu =]0,1[$. Figures 4.38-4.42 below illustrate the effects of applying momentum to BPNN training. All the experimental results are based on modified BPNN training with parameters of momentum for 0.0, 0.2, 0.5 and 0.9, learning rate at 1.0, slope parameter at 1.0 and MSE error limit at 0.001. From the figures, the most efficient value of momentum varies as the *n* changes. The optimum momentum can only be determined via the experimental testing.

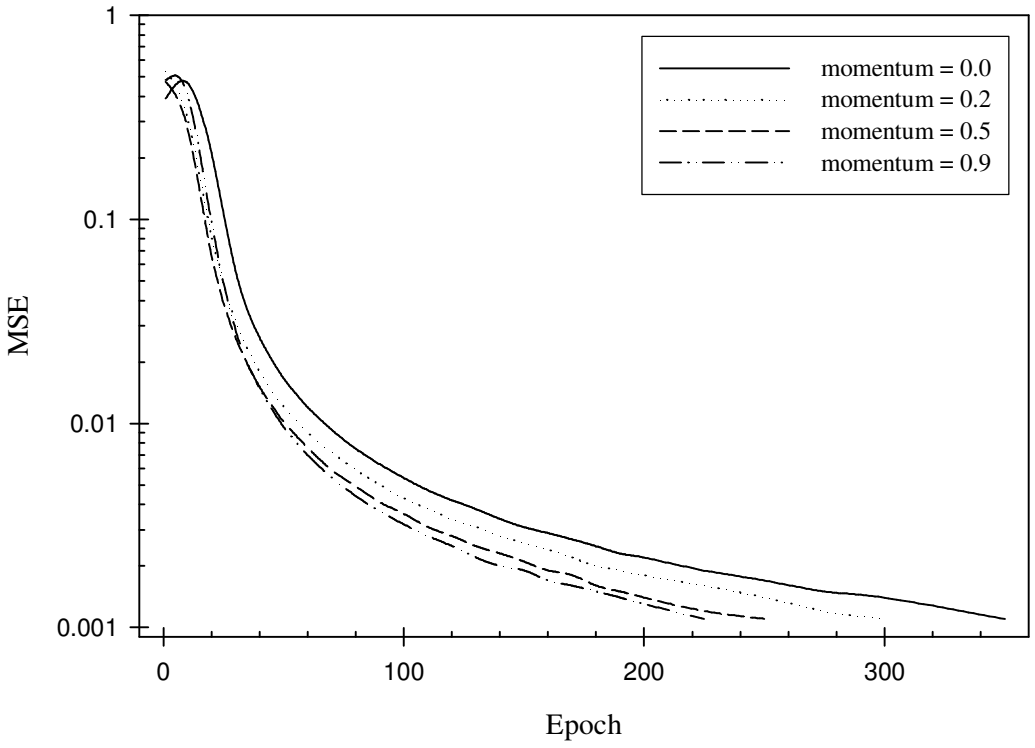


Figure 4.38 Required epoch for FCN-4 (momentum = 0.0, 0.2, 0.5, 0.9)

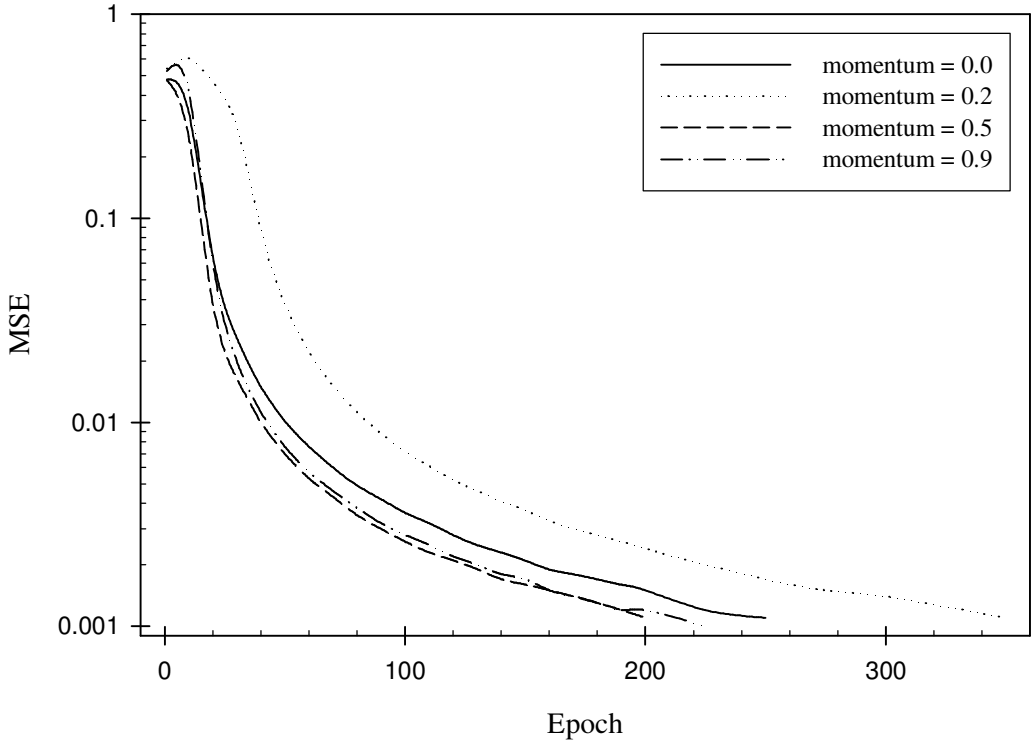


Figure 4.39 Required epoch for FCN-5 (momentum = 0.0, 0.2, 0.5, 0.9)

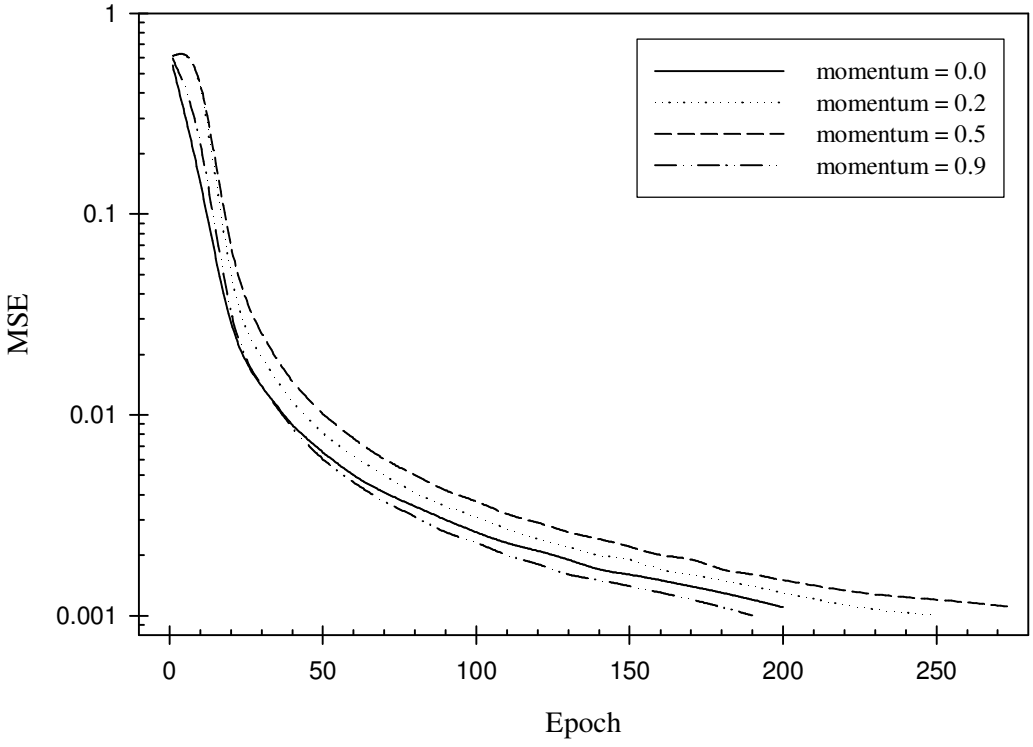


Figure 4.40 Required epoch for FCN-6 (momentum = 0.0, 0.2, 0.5, 0.9)

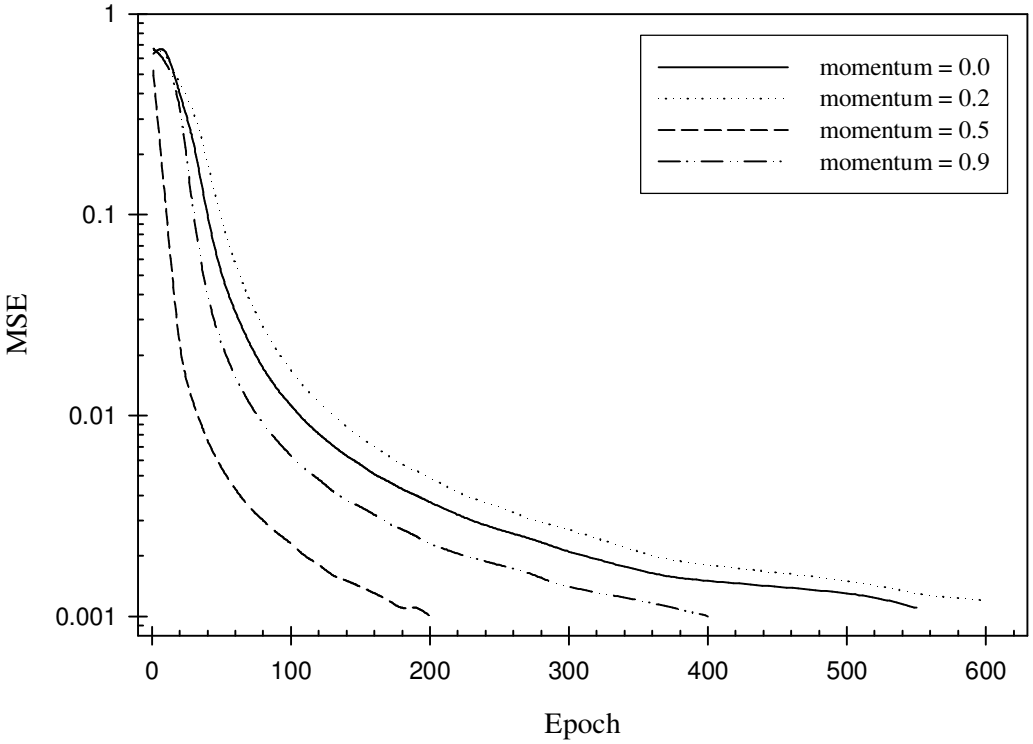


Figure 4.41 Required epoch for FCN-7 (momentum = 0.0, 0.2, 0.5, 0.9)

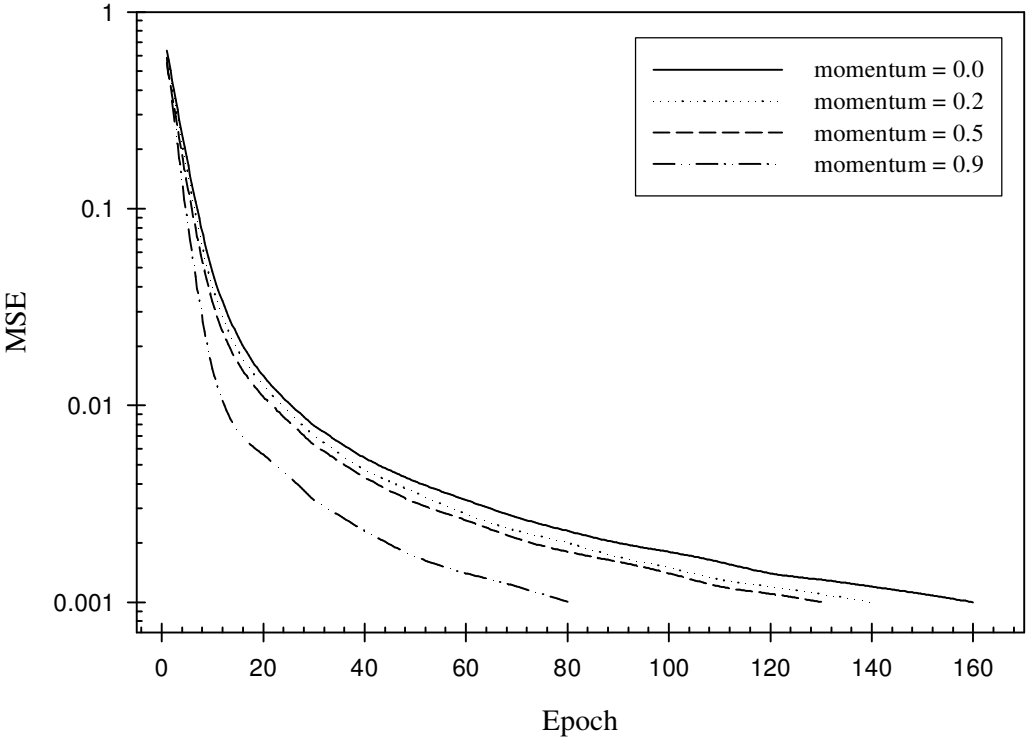


Figure 4.42 Required epoch for FCN-10 (momentum = 0.0, 0.2, 0.5, 0.9)

4.6.4 Effects of Learning Rate to BPNN training

Instead of momentum, another parameter in the standard BPNN can be modified to gain faster BPNN training. This is the learning rate, α, which accounts for the weight corrections as well as the bias corrections in the error space of the neural networks. For standard BPNN, $\alpha = 1$. This learning rate can be modified to be smaller than 1 or larger than 1. However, even though having learning rate more than 1 will allow faster weight and bias adjustment, it will cause the oscillation problem that restricts the BPNN from reaching either local or global minimum. Hence, as the BPNN training proceeds, learning rate is normally set high and reduced gradually in the interval of $(0 < \alpha < 1)$ together with momentum μ. The adaptive learning rate in this proposed ANN based BAP depends on the mean squared error, MSE. Figures 4.43-4.47 below show the effects of adaptive learning rate. All the experimental results are based on modified BPNN training with parameters of momentum at 0.0, learning rates for 0.5, 0.8, 1.0 and 2.0, slope parameter at 1.0 and MSE error limit at 0.001. From the figures, the adaptive learning rate gives a faster speed of convergence than the fixed learning rate.

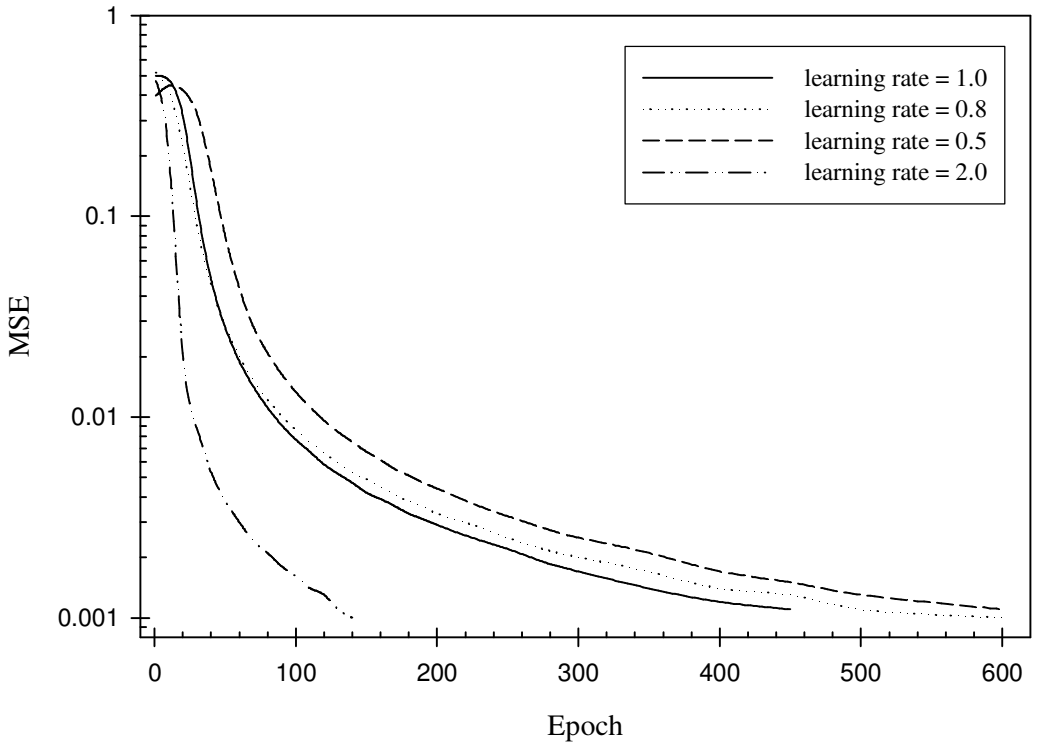


Figure 4.43 Required epoch for FCN-4 (learning rate = 1.0, 0.8, 0.5, 2.0)

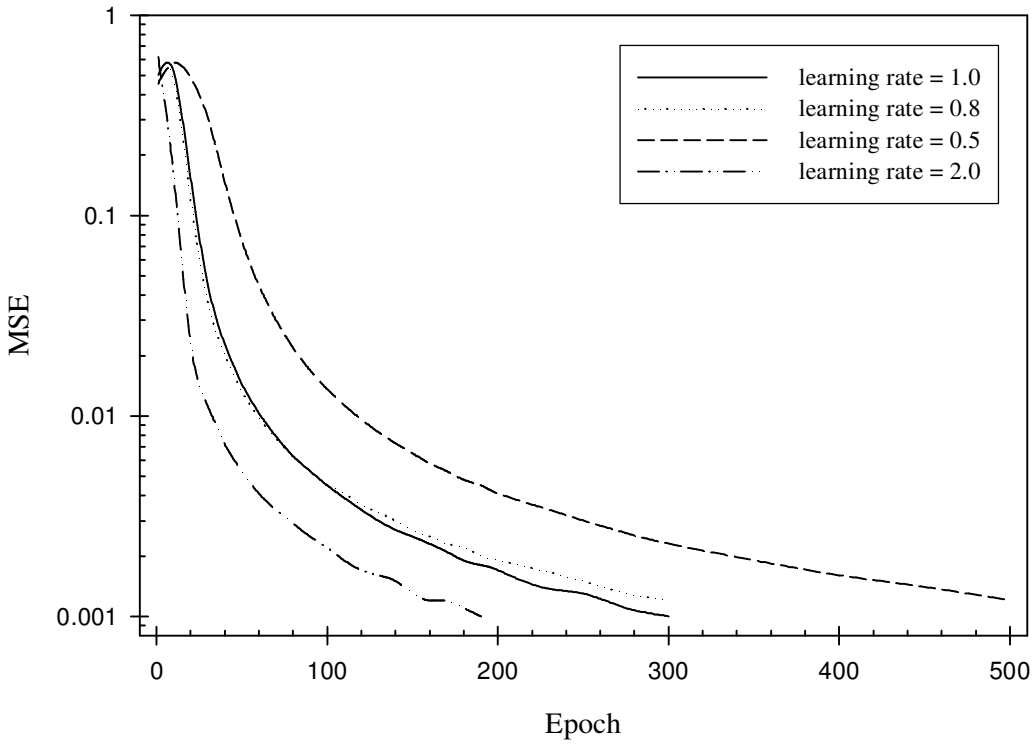


Figure 4.44 Required epoch for FCN-5 (learning rate = 1.0, 0.8, 0.5, 2.0)

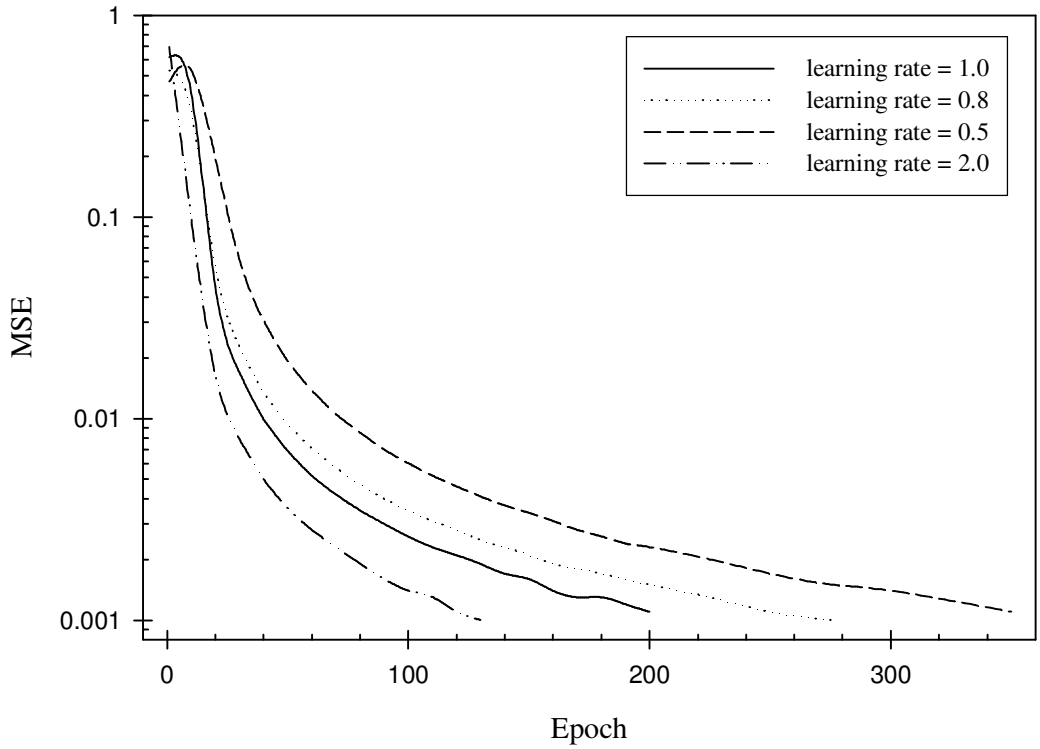


Figure 4.45 Required epoch for FCN-6 (learning rate = 1.0, 0.8, 0.5, 2.0)

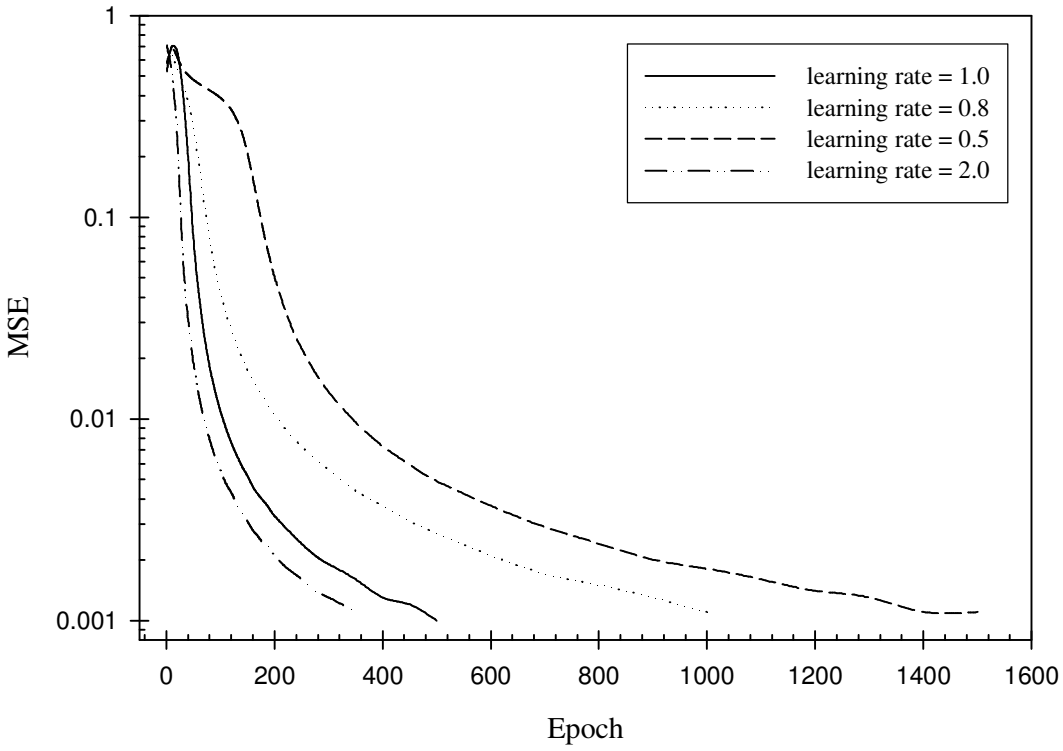


Figure 4.46 Required epoch for FCN-7 (learning rate = 1.0, 0.8, 0.5, 2.0)

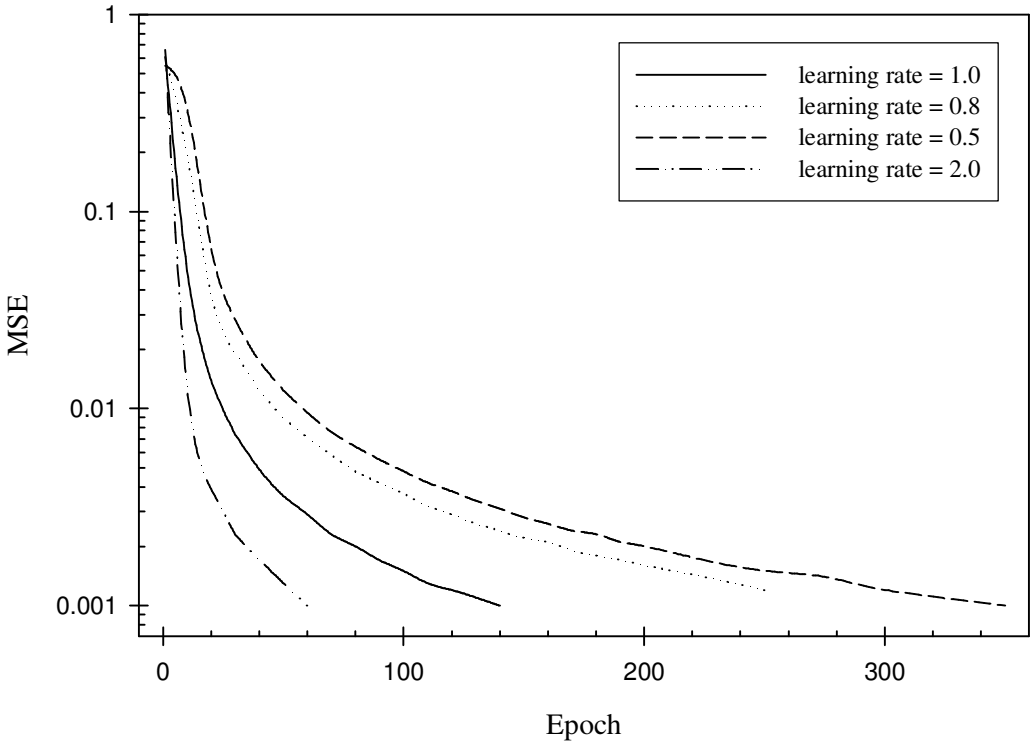


Figure 4.47 Required epoch for FCN-10 (learning rate = 1.0, 0.8, 0.5, 2.0)

4.6.5 Effects of Slope Parameter to BPNN training

Besides the momentum and adaptive learning rate, there is another parameter that could be modified to gain faster speed of learning. This parameter is slope parameter, σ. It is adjusted empirically to determine the optimal value for BPNN training. The effects of slope parameter are illustrated in the Figures 4.48-4.52 below. All the experimental results are based on modified BPNN training with parameters of momentum at 0.0, learning rate at 1.0, slope parameters for 1.0, 0.5, 2.0 and 5.0, and MSE error limit at 0.001. From the figures, a higher value of slope parameter increases the speed of convergence. However, there is a critical maximum value of slope parameter. Once the critical value is exceeded, the oscillation problem emerges, and the BPNN training will not reach any global nor local minimum.

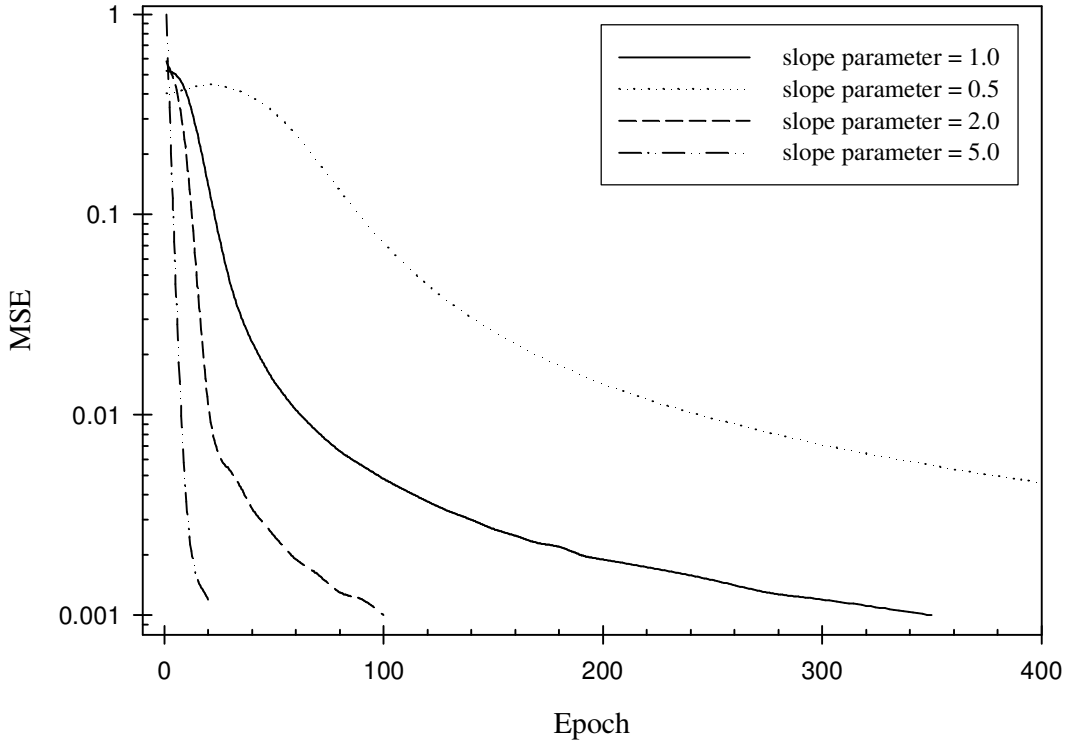


Figure 4.48 Required epoch for FCN-4 (slope parameter = 1.0, 0.5, 2.0, 5.0)

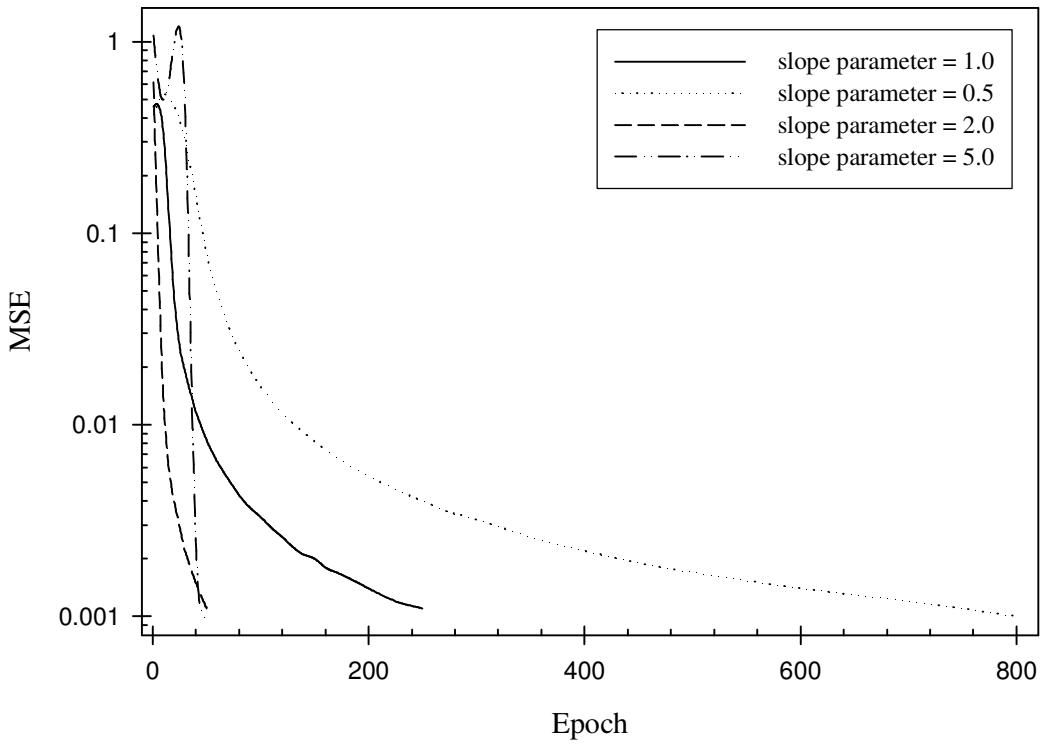


Figure 4.49 Required epoch for FCN-5 (slope parameter = 1.0, 0.5, 2.0, 5.0)

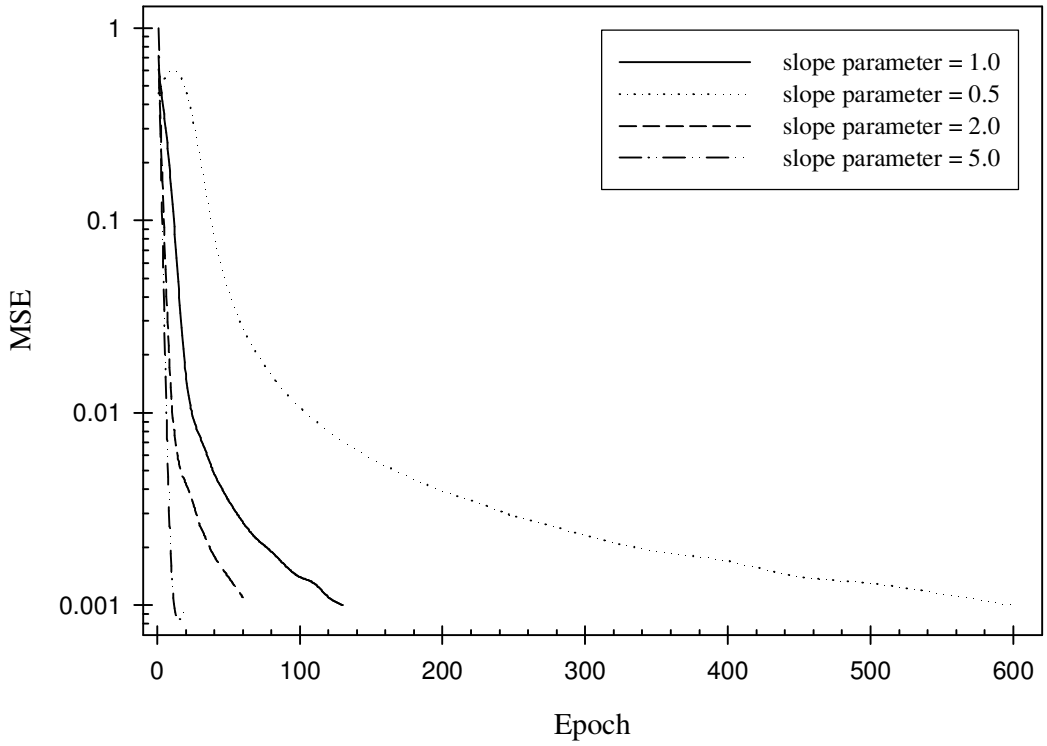


Figure 4.50 Required epoch for FCN-6 (slope parameter = 1.0, 0.5, 2.0, 5.0)

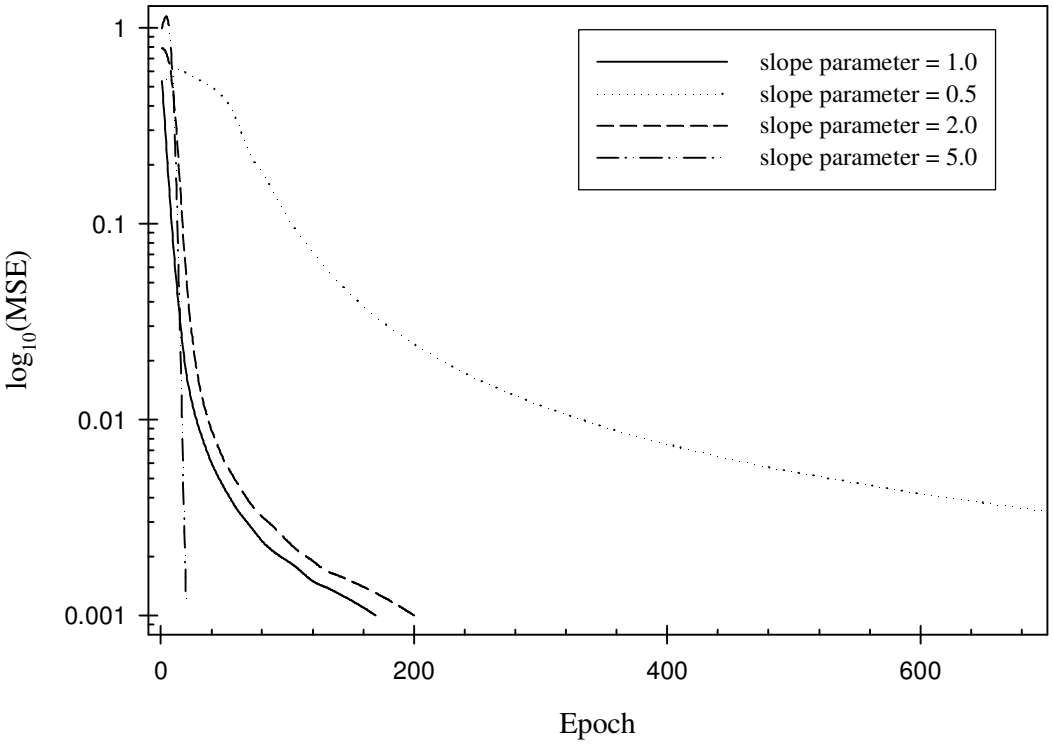


Figure 4.51 Required epoch for FCN-7 (slope parameter = 1.0, 0.5, 2.0, 5.0)

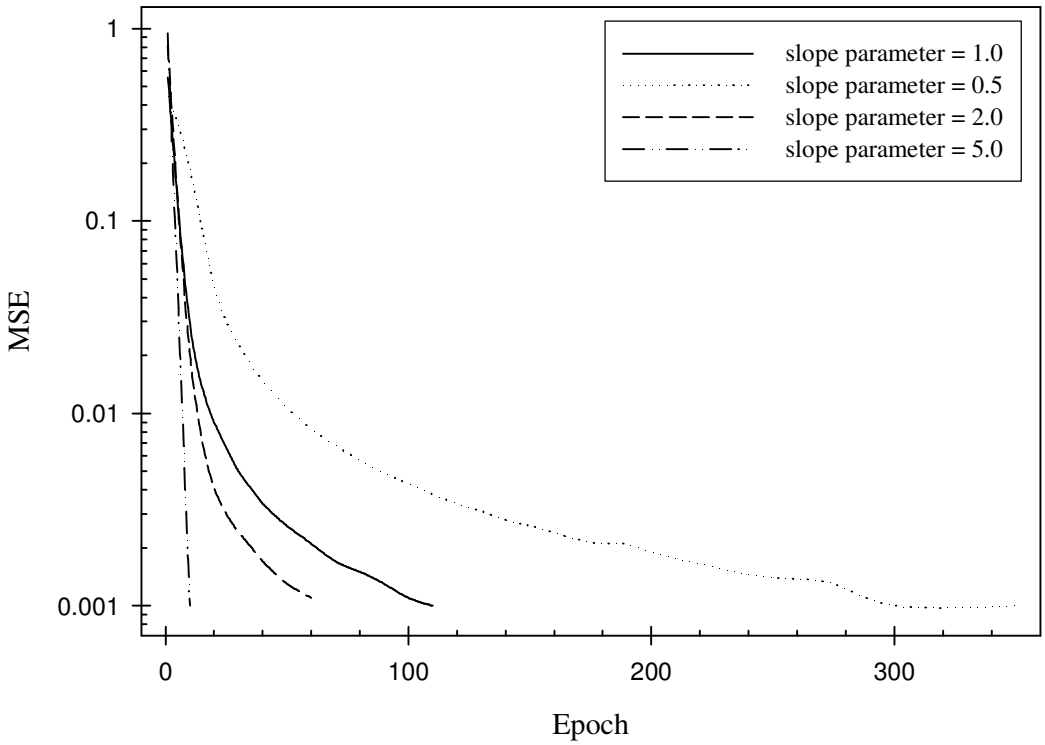


Figure 4.52 Required epoch for FCN-10 (slope parameter = 1.0, 0.5, 2.0, 5.0)

4.6.6 Performance of Modified BPNN

Combining the adjustments on momentum, learning rate and slope parameter, an optimized modified BPNN can be found for the ANN based BAP of FCN-*n*. Various combinations of modified BPNN training are tested. All the results are from modified BPNN with parameters of momentum (0.0, 0.2, 0.5 and 0.9), learning rate (0.2, 0.5, 0.8 and 1.0), slope parameter at 1.0, and MSE error limit at 0.001.

Figure 4.53 shows the modified BPNN with slope parameter at 2.0 for all the combinations of learning rate (α = 1.0 and 0.5) and momentum (μ = 0.9, 0.5 and 0.2). Figures 4.54-4.56 (n = 4) and Figures 4.60-4.62 (n=10) show the modified BPNN with fixed learning rate (α = 0.8, 0.5 or 0.2) corresponding to varying momentum (μ = 0.9, 0.5, 0.2 and 0.0) at slope parameter of 1.0 and MSE of 0.001. Figures 4.57-4.59 (n = 4) and Figures 4.63-4.65 (n=10) show the modified BPNN with fixed momentum (μ = 0.9, 0.5 or 0.2) corresponding to varying learning rate (α = 1.0, 0.8, 0.5, 0.2) at slope parameter of 1.0 and MSE of 0.001. From the figures, a good combination of modified BPNN is to have high value of learning rate but a corresponding small value of momentum.

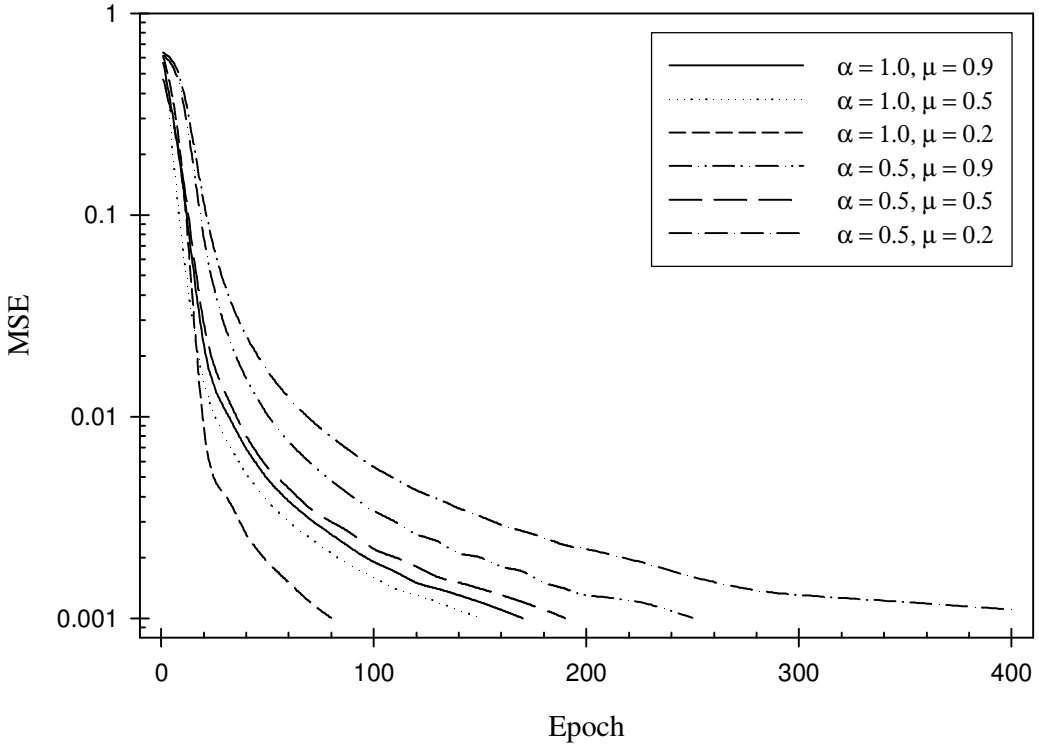


Figure 4.53 Required epoch for FCN-4 (slope parameter = 2.0; momentum = 0.2, 0.5 and 0.9; learning rate = 1.0 and 0.5)

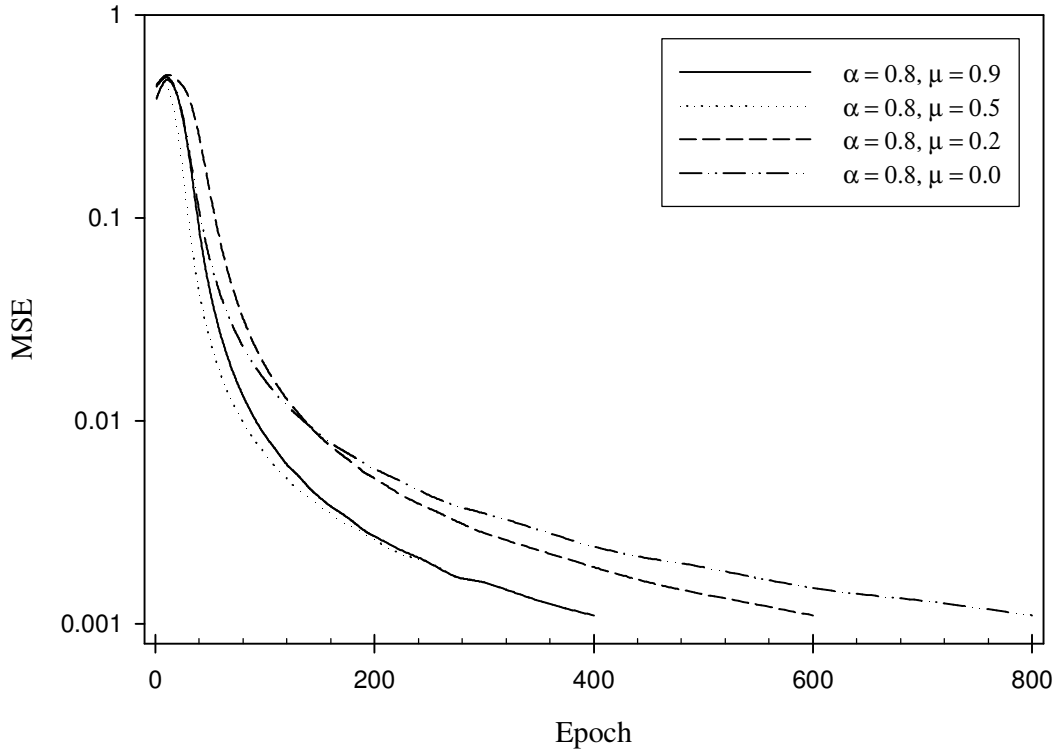


Figure 4.54 Required epoch for FCN-4 (learning rate = 0.8; momentum = 0.9, 0.5, 0.2, 0.0)

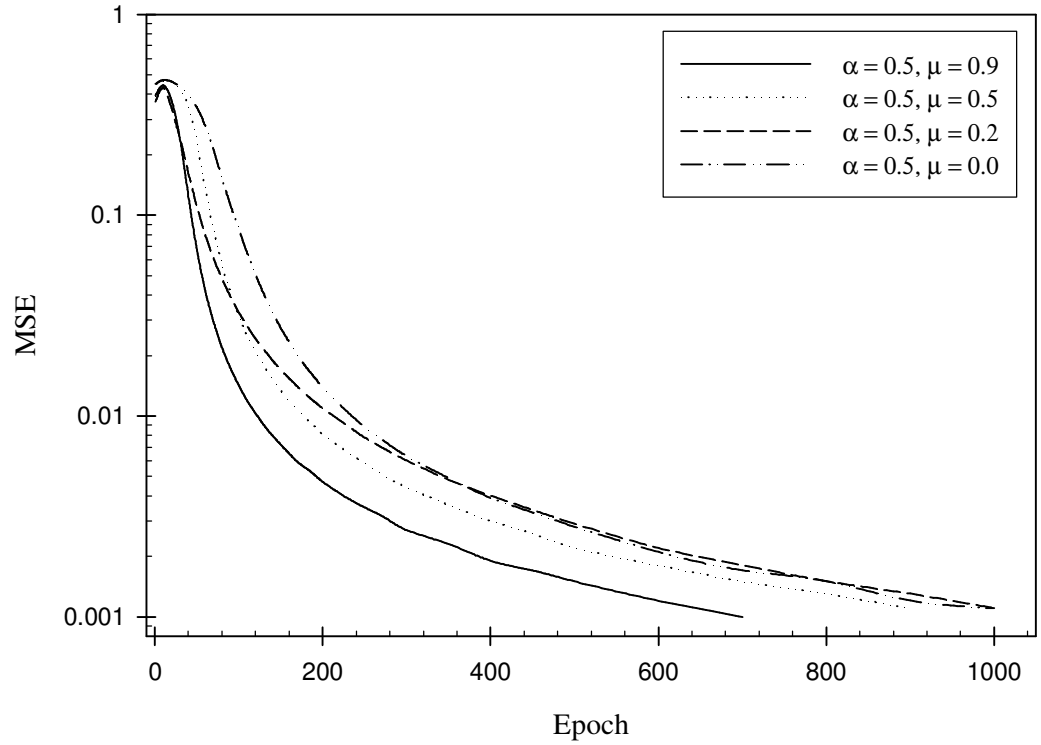


Figure 4.55 Required epoch for FCN-4 (learning rate = 0.5; momentum = 0.9, 0.5, 0.2, 0.0)

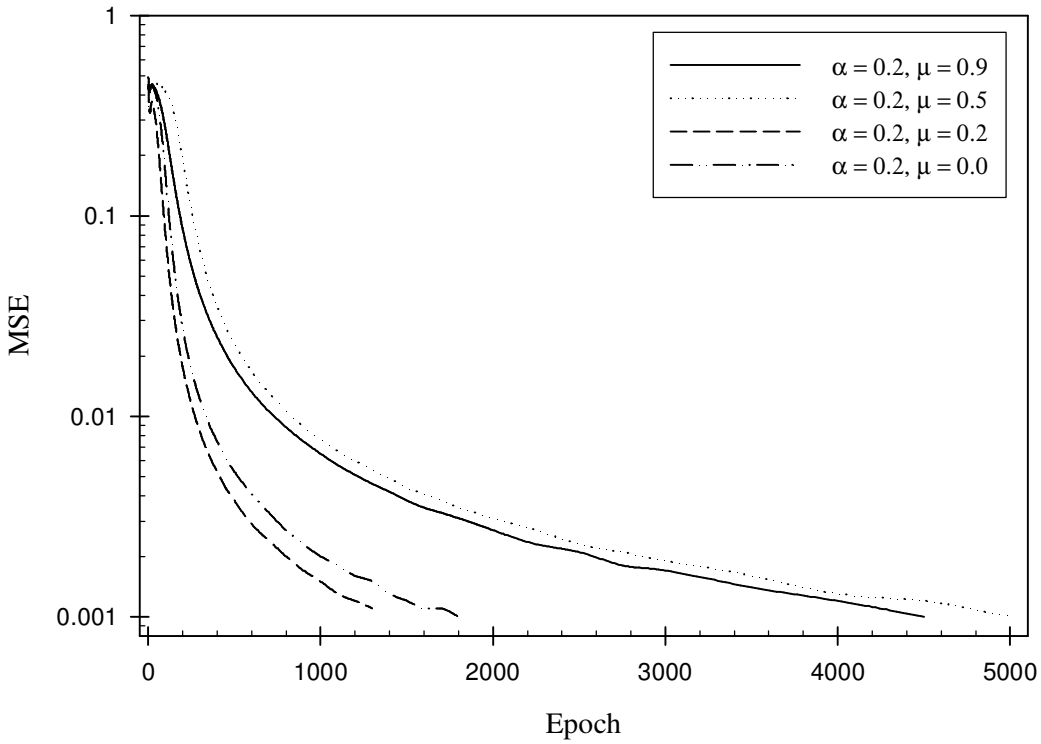


Figure 4.56 Required epoch for FCN-4 (learning rate = 0.2; momentum = 0.9, 0.5, 0.2, 0.0)

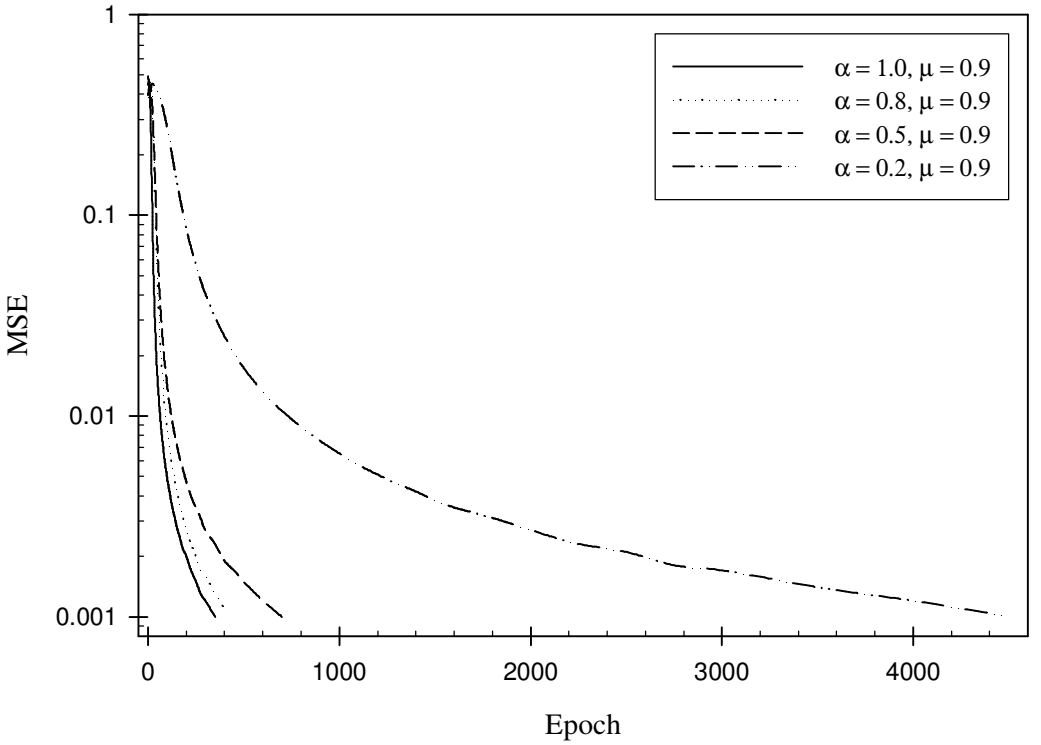


Figure 4.57 Required epoch for FCN-4 (momentum = 0.9; learning rate = 1.0, 0.8, 0.5, 0.2)

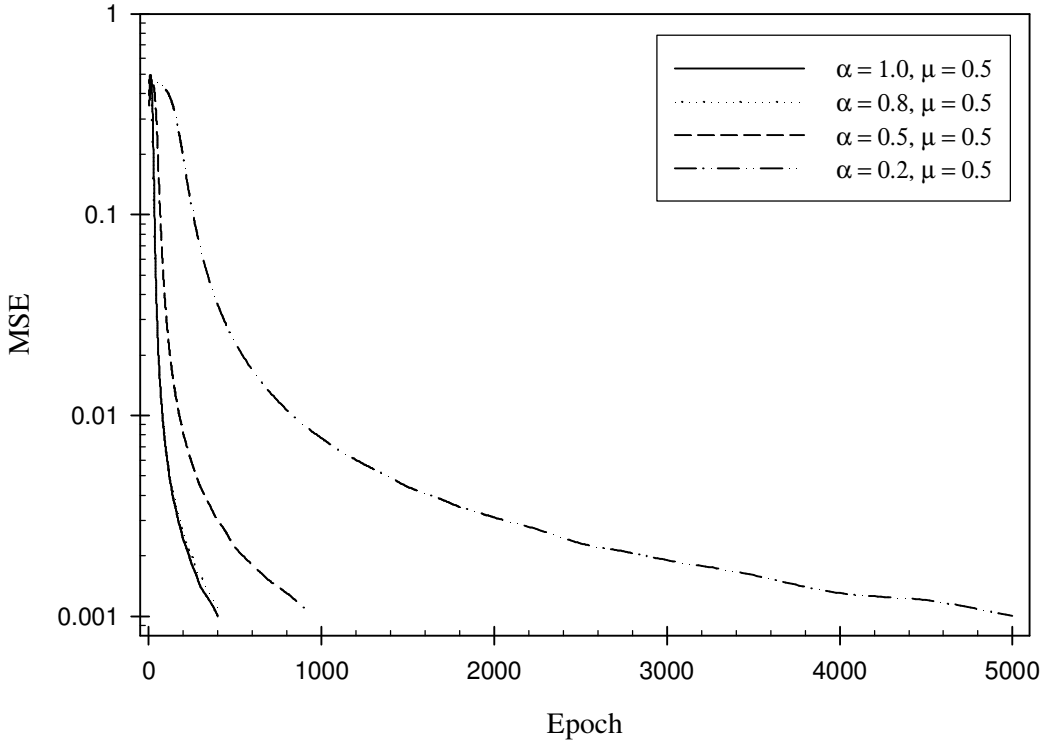


Figure 4.58 Required epoch for FCN-4 (momentum = 0.5; learning rate = 1.0, 0.8, 0.5, 0.2)

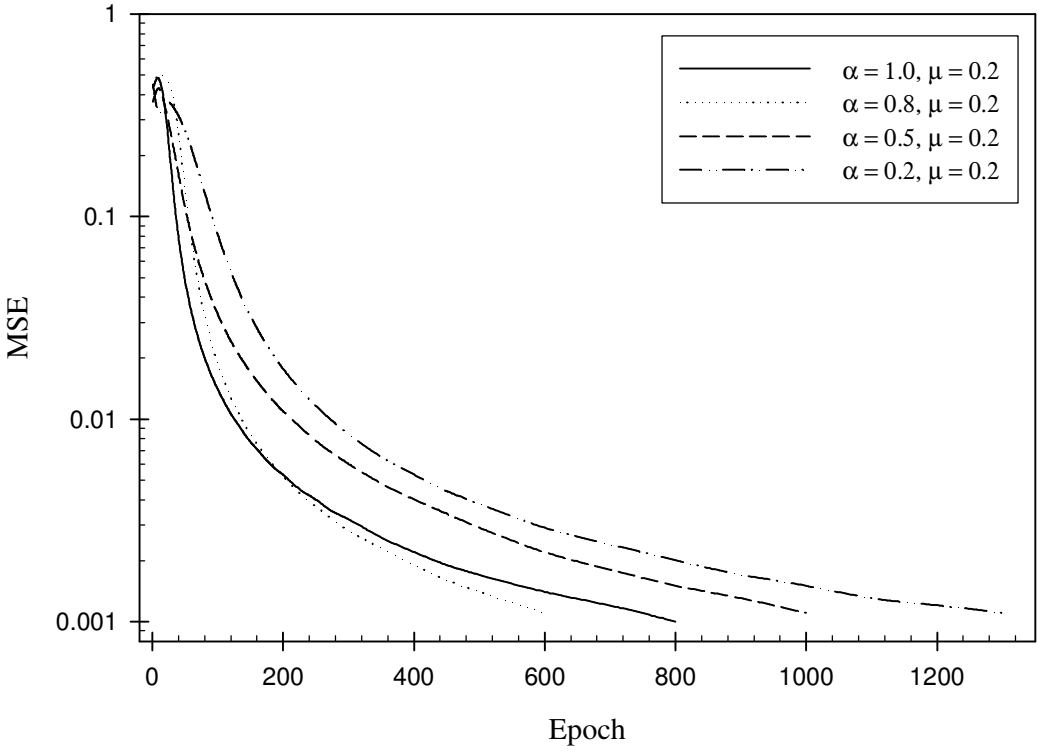


Figure 4.59 Required epoch for FCN-4 (momentum = 0.2; learning rate = 1.0, 0.8, 0.5, 0.2)

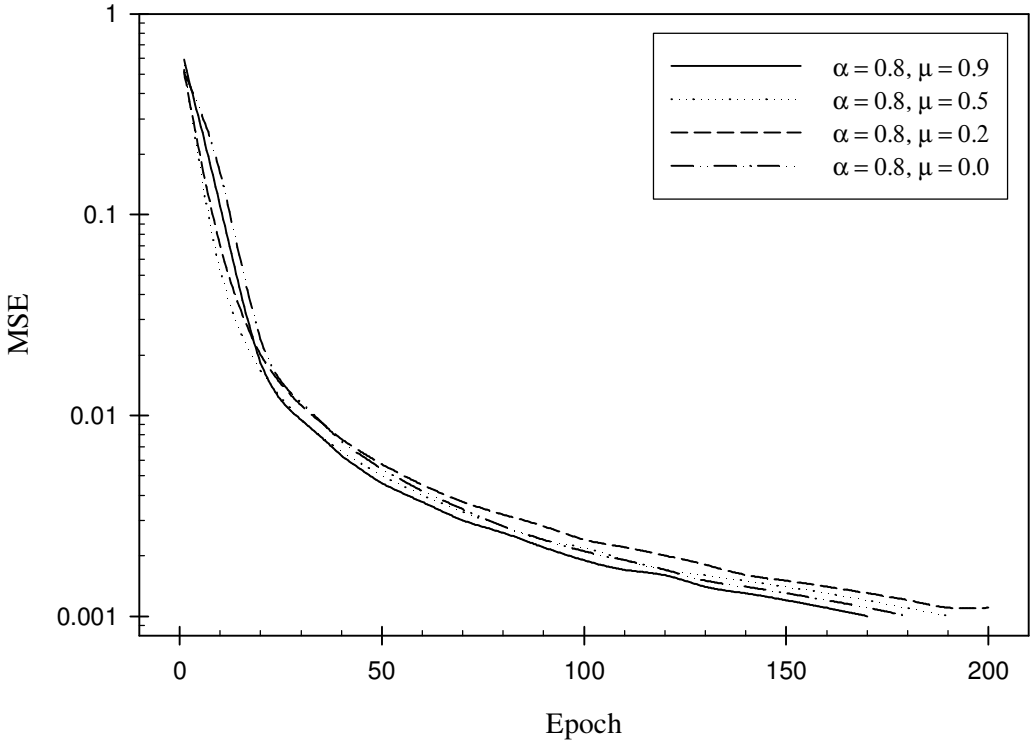


Figure 4.60 Required epoch for FCN-10 (learning rate = 0.8; momentum = 0.9, 0.5, 0.2, 0.0)

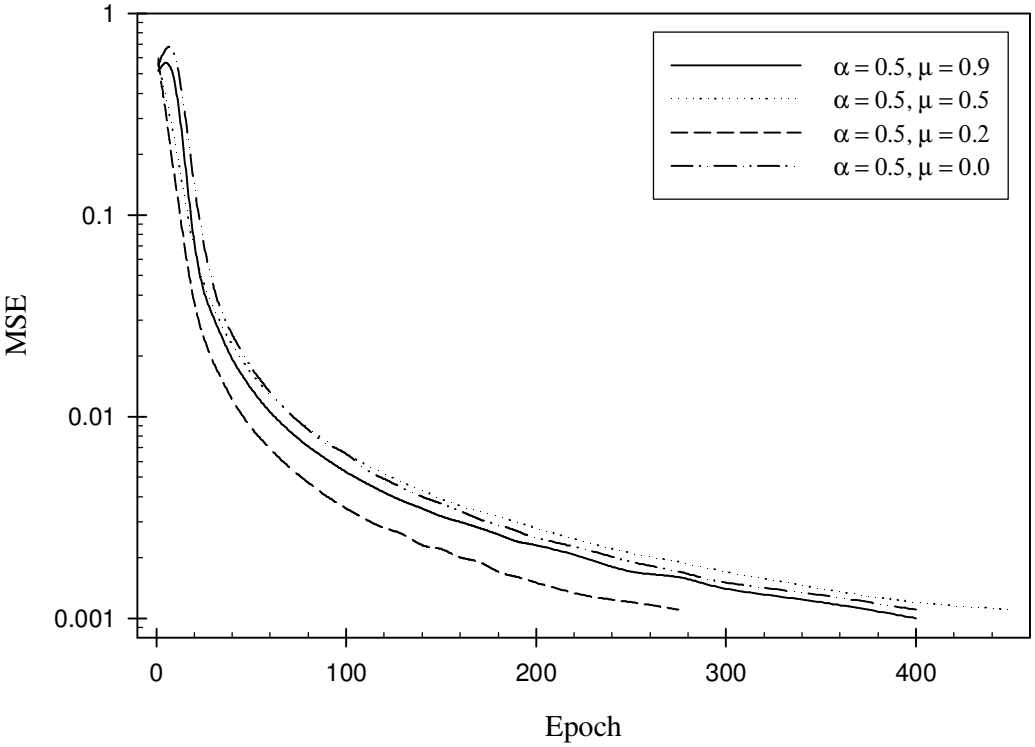


Figure 4.61 Required epoch for FCN-10 (learning rate = 0.5; momentum = 0.9, 0.5, 0.2, 0.0)

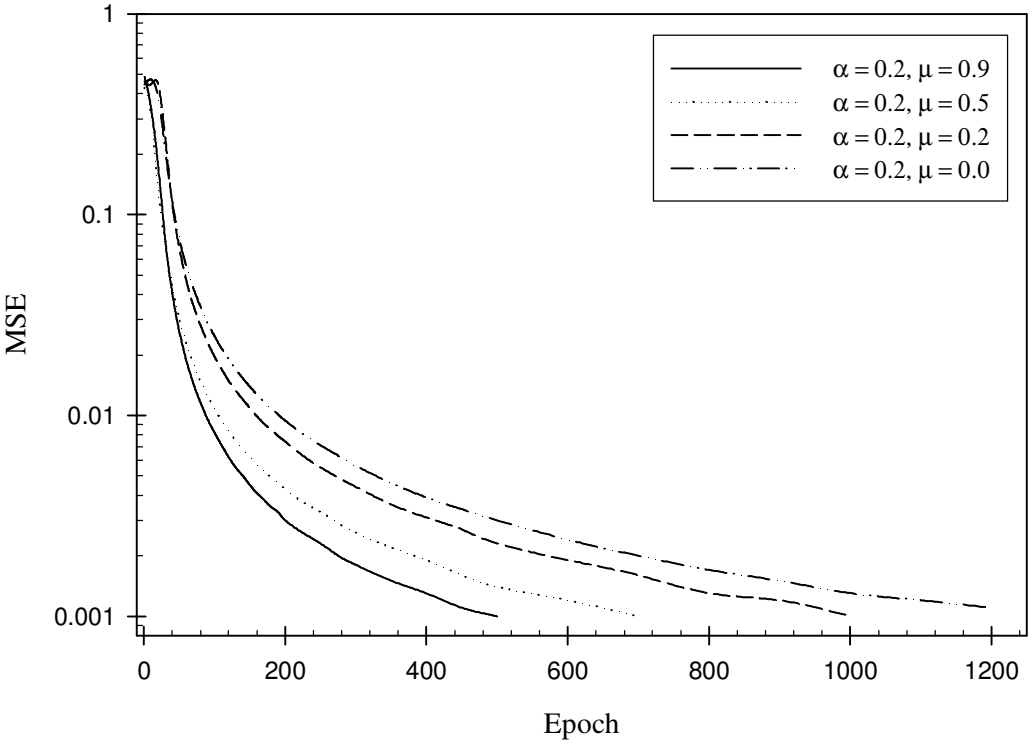


Figure 4.62 Required epoch for FCN-10 (learning rate = 0.2; momentum = 0.9, 0.5, 0.2, 0.0)

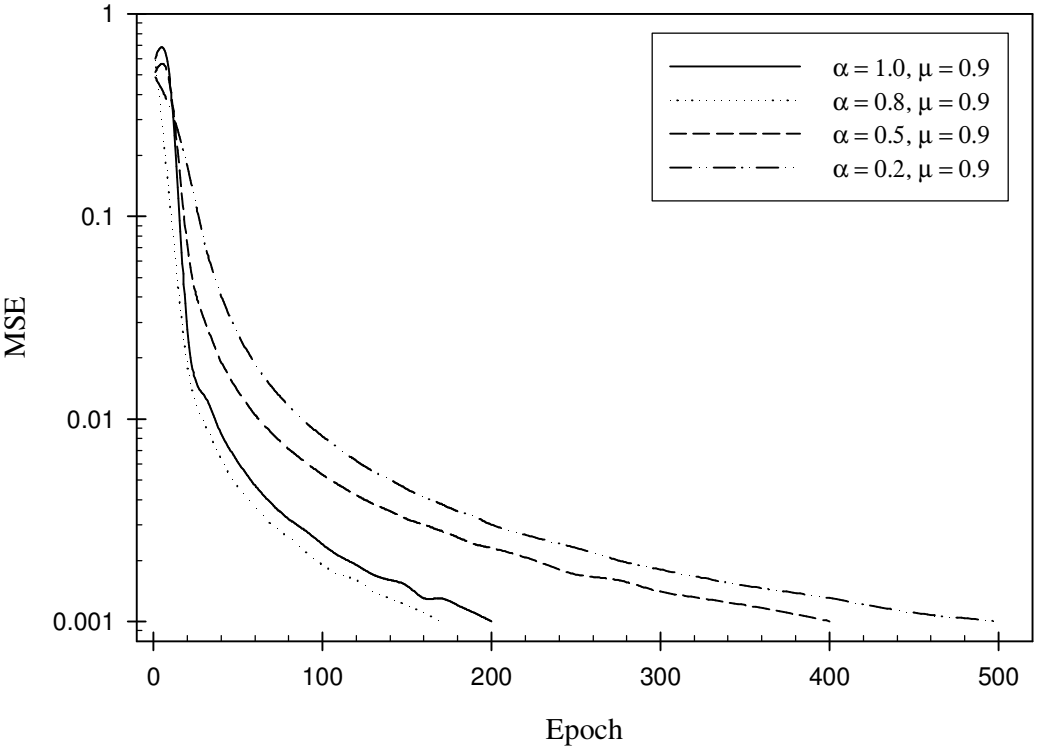


Figure 4.63 Required epoch for FCN-10 (momentum = 0.9; learning rate = 1.0, 0.8, 0.5, 0.2)

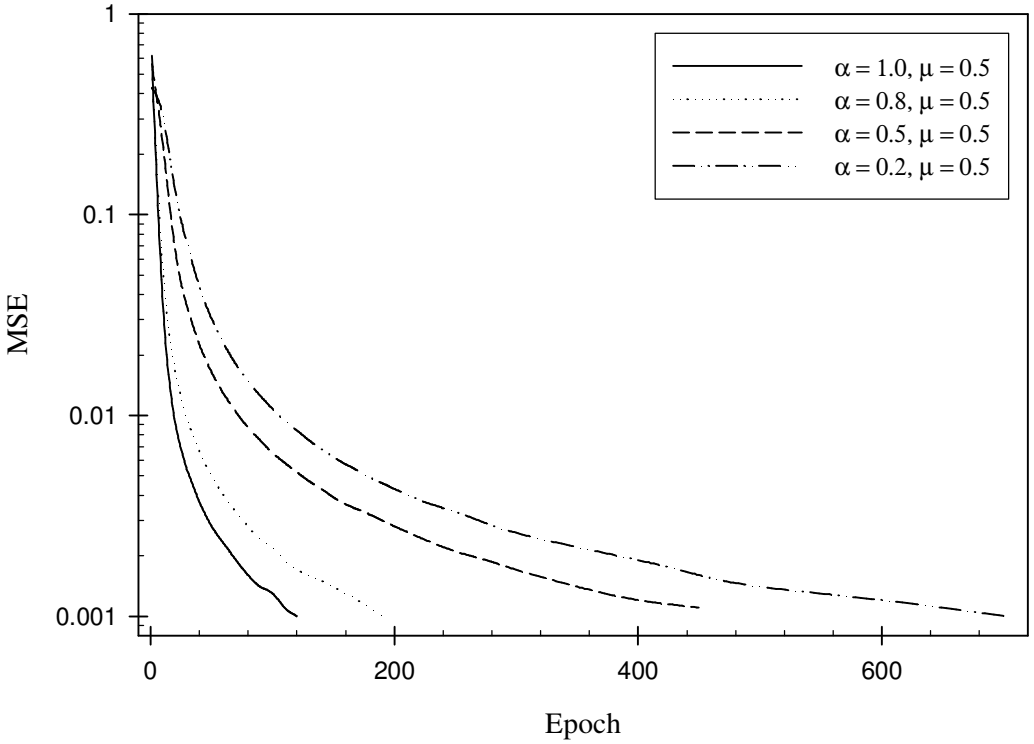


Figure 4.64 Required epoch for FCN-10 (momentum = 0.5; learning rate = 1.0, 0.8, 0.5, 0.2)

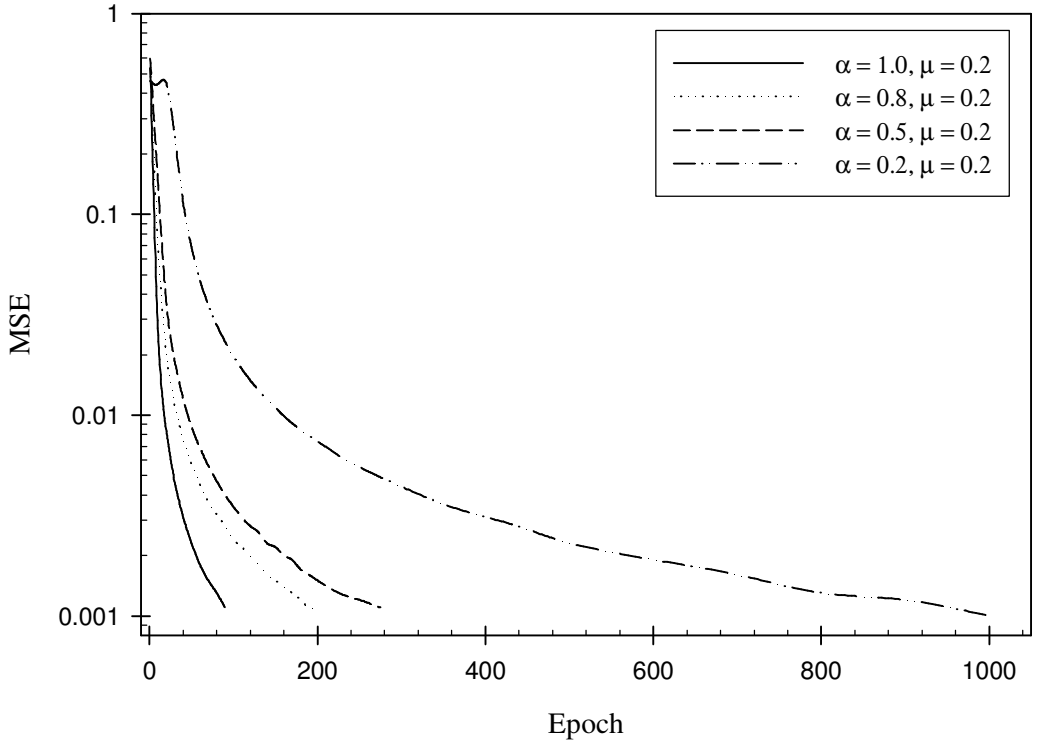


Figure 4.65 Required epoch for FCN-10 (momentum = 0.2; learning rate = 1.0, 0.8, 0.5, 0.2)

4.7 COMPLEXITY OF ANN BASED BAP

To compare the complexity of ANN based BAP with the traditional BAP, we need to recall the total number of message being exchanged to achieve Byzantine Agreement. For traditional BAP, the number of rounds of message exchange, $Rnd\_exch_{Trad.}$, and the total number of message being exchanged, $Msg_{Trad.}$, to achieve Byzantine Agreement, are given as below:

$$Rnd\_exch_{Trad.} = m + 1 \tag{4.45}$$

$$Msg_{Trad.} = n - 1 + \sum_{k=1}^{m} \frac{(n-1)!}{(n-k-2)!} \tag{4.46}$$

$$\text{Complexity of traditional BAP} = O(n^{m+1}) \tag{4.47}$$

where n is the total number of nodes,

and m is the number of faulty nodes in a malicious network.

It can be derived from Equation (4.46) to get Equation (4.47) that the worst case upper bound for traditional BAP is given by $O(n^{m+1})$ where m is the floor function of $(n\text{-}1)/3$. Hence the complexity of the BGP using traditional BAP increases exponentially as the number of nodes gets bigger.

Meanwhile, for the proposed ANN based BAP, the number of rounds of message exchange, Rnd_exch_{ANN}, and the total number of message being exchanged, Msg_{ANN}, to achieve Byzantine Agreement, are given in Equations (4.48) and (4.49). Notice that only a maximum of three rounds is required for the number of rounds of message exchange. This allows a great reduction in memory space requirement. From Equation (4.49), the complexity of ANN based BAP is derived to be $O(n^3)$ as in Equation (4.50). The total number of exchanged messages for traditional BAP and ANN based BAP are compared in Table 4.3.

$$Rnd\_exch_{ANN} = 3 \tag{4.48}$$

$$Msg_{ANN} = (n - 1)^3 \tag{4.49}$$

$$\text{Complexity of ANN based BAP} = O(n^3) \tag{4.50}$$

Table 4.3 Total number of exchanged messages, *Msg*, for traditional BAP and ANN based BAP for $n = 4, 5, 6, \ldots, 25$

n	m	Traditional BAP	ANN based BAP
4	1	9	27
5	1	16	64
6	1	25	125
7	2	156	216
8	2	259	343
9	2	400	512
10	3	3609	729
11	3	5860	1000
12	3	9031	1331
13	4	108384	1728
14	4	173485	2197
15	4	266644	2744
16	5	3999675	3375
17	5	6337216	4096
18	5	9714769	4913
19	6	174865860	5832
20	6	274985119	6859
21	6	420592000	8000
22	7	8832432021	9261
23	7	1389734884	10648
24	7	21081995155	12167
25	8	505967883744	13824

In Figure 4.66, it shows the graph of the total number of exchanged message (*Msg*) against the number of processors (*n*) for the traditional BAP and ANN based BAP using standard or traditional BPNN. From the figure, for small number of nodes in a distributed network, traditional BAP is found to be more efficient than ANN based BAP. However, as $n \geq 10$, ANN based BAP becomes more efficient.

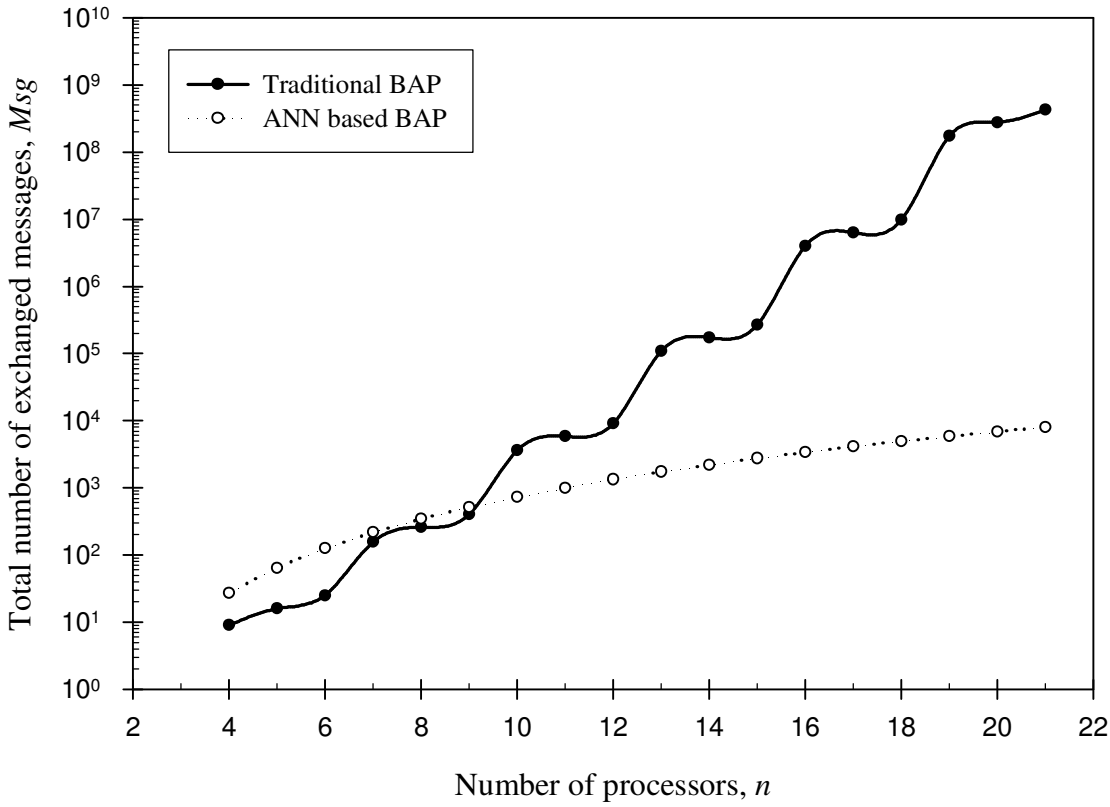


Figure 4.66 Total number of exchanged messages, *Msg*, versus number of processors, *n*, for traditional BAP and ANN based BAP

Hence once the artificial neural network is trained up, we can conclude that ANN based BAP is much more suitable to a large distributed network as compared to the traditional BAP. This is because ANN based BAP is of the complexity of $O(n^3)$ and the traditional BAP is of the complexity of $O(n^{m+1})$. Hence as the number of faulty nodes, *m*, is equal or more than three, the efficiency of ANN based BAP will overtake the traditional BAP.

4.8 ADVANTAGES OF ANN BASED BAP

Inherently, the ANN based BAP has some superior advantages from the natural properties of neural network. The advantages given by the three natural properties of ANN are explained in Section 3.2.3 for general neural network applications. To the ANN based BAP specifically, these three properties listed below applied as well.

i. Storing the information and knowledge within the weights among the interconnected nodes.

ii. Massively parallel distributed computing structures.

iii. Learning capability and generalization.

These three properties enable the neural network to solve the complicated problem of BGP. The first property allows a greatly reduced requirement for memory space of computation. The second property makes the parallel processing ability at each node to become reality. The third property gives the flexible learning capability to each node to encounter the versatile and dynamic Byzantine environment. These three main advantages make the ANN based BAP to perform better than the traditional BAP.

Instead of the main advantages, neural network introduces some other advantages to the BGP. Neural network is capable of simulating the non-linear function of BGP. It is also capable of mapping the input vectors of the received messages to the output vectors of the decision to be made up. In addition, the coupling of input-output mapping capability and the ANN adaptivity enables ANN based BAP to have adaptive pattern classification ability over the vectors of received messages. The adaptivity allows the neural network to be re-trained when there is a minor variation in the operating environmental conditions. This is important and useful to handle the BGP, which is changing with time in a statistical situation.

Furthermore, the advance of hardware technologies allows the ANN to have VLSI implementability (Mead & Conway, 1980). This allows the ANN based BAP to be applicable for real-time application by embedding the neural network into

every processor within the distributed system. Because of the ANN natural distributed architecture, it will have fault-tolerant property when ANN is hardware implemented (Bolt, 1992). Hence, ANN based BAP performs a graceful degradable failure rather than an immediate catastrophic failure.

4.9 CONCLUSIONS

Artificial neural network based Byzantine Agreement Protocol (ANN based BAP) is discussed in details throughout Chapter 4. The background of ANN is brought forward in Section 4.2 and the gradient descent based back propagation neural network (BPNN) is in Section 4.3. A variance from standard BPNN, which is called modified BPNN, is included in Section 4.4.

The proposed ANN based BAP is analyzed theoretically step by step under Section 4.5. It is divided into five stages for easier understanding. The stages are initialization stage, message exchange stage, training stage, application stage and compromise stage. Under Section 4.6, the BPNN learning algorithm of ANN based BAP is tested. The neural network model of ANN based BAP is tested for its capability of convergence under BPNN by passing through a series of experiments with changing parameters: bias, weight initialization, momentum, learning rate, slope parameter, and MSE error limit. The improvements of the convergence speed of ANN based BAP using the advanced techniques of Nguyen-Widrow initialization and modified BPNN are documented in Lee and Ewe (2003c).

At last, both the complexity and advantages of ANN based BAP are compared with traditional BAP in Section 4.7 and 4.8 respectively. For a malicious network having n number of nodes, the ANN based BAP is more efficient than traditional BAP when $n \geq 10$. This is because the total number of exchanged messages in ANN based BAP will be a big amount less than traditional BAP after the threshold of $n = 10$. The advantages of ANN based BAP are its ability to store memory in weights, and hence reduces the huge requirement for memory space in traditional BAP, its massively parallel distributed computing structures and its learning capability over a dynamic environment.

CHAPTER 5: ANALYSIS OF MESSAGE EXCHANGE MATRIX

5.1 INTRODUCTION

In Section 4.5, the ANN based BAP is implemented over a 4-processor distributed system. For a 4-node fully connected network (FCN) to be solved by the ANN based BAP, it needs to pass through all the five stages: initialization stage, message exchange stage, training stage, application stage and compromise stage. In the message exchange stage, a matrix is formed from the rounds of message exchange. This matrix, which is known as message exchange matrix (MEM), will be analyzed in this chapter.

5.2 FORMATION OF MESSAGE EXCHANGE MATRIX

A message exchange matrix (MEM) is formed in the message exchange stage of the ANN based BAP. For a n-node network, the re-arrangement of exchanged messages through three rounds of message exchange forms a MEM with the dimensions of $(n\text{-}1)*(n\text{-}1)$. The constructed MEM is used for both the training set and the testing set for the artificial neural network. To show how a MEM is formed, the general case for n-processor distributed system together with two specific cases are adopted here, i.e. the $n = 4$ and 7.

For the general case of an n-processor distributed computer system, the system is simulated into a graph of n-node of fully connected network (FCN). In the first round, the commander node broadcasts $(n\text{-}1)$ messages to all the other lieutenant nodes. In the second round, each lieutenant node sends the received message to the other $(n\text{-}2)$ lieutenant nodes. At the end of second round, each lieutenant node holds $(n\text{-}1)$ messages from the other nodes. In the third round or last round, each lieutenant node delivers $(n\text{-}1)$ messages to the other $(n\text{-}2)$ lieutenant nodes. Therefore, the total number of exchanged messages to be received by each lieutenant node, Msg_{ANN_node} is $(n\text{-}1)^2$ as in Equation (5.1). For total number of exchanged messages within the

neural network, Msg_{ANN}, it is given by Equation (4.41). For easy reference, it is repeated here as Equation (5.2).

$$Msg_{ANN\_node} = 1+(n-2)+(n-1)*(n-2)$$

$$= (n-1)^2 \tag{5.1}$$

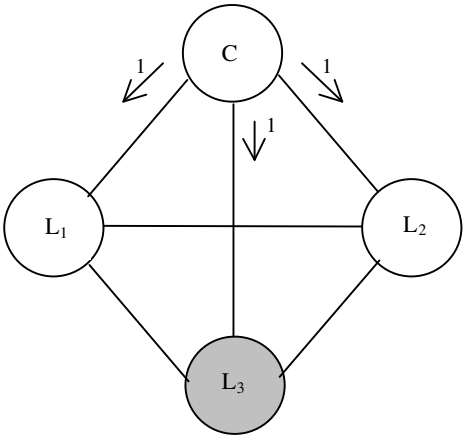
$$Msg_{ANN} = (n-1)*(n-1)^2$$

$$= (n-1)^3 \tag{5.2}$$

At each of the $(n-1)$ lieutenant node or processor, there are $(n-1)^2$ received messages. At the same time, it shall be aware that at each lieutenant node, there is an embedded ANN. The received $(n-1)^2$ messages at each node are used as both the training set and testing set for the embedded neural network. These $(n-1)^2$ messages are arranged into a square matrix MEM in a manner that the row dimension represents the sender node and the column dimension represents the receiver node. To have better understanding of how a MEM is formed, follow two of the examples as below: $n = 4$ and $n = 7$.

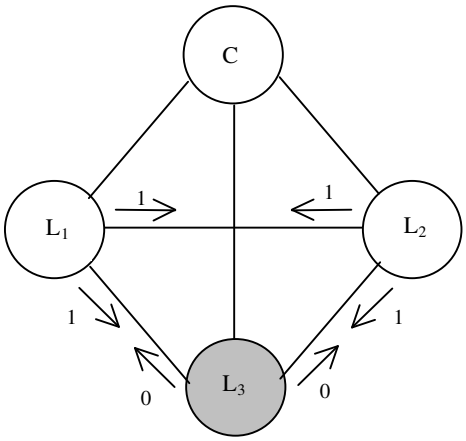
To illustrate the formation of MEM during the message exchange stage of a 4-processor distributed system, Figures 4.16, 4.17 and 4.18 are integrated into Figure 5.1 and the corresponding contents of MEM at each round of message exchange are shown in Figure 5.2. In Figure 5.1, it is a 4-node FCN having one commander node C and three lieutenant nodes L_1, L_2 and L_3. For the worst case, there exists one faulty node among the three lieutenant nodes. Hence, we have three faulty situations here, i.e. either L_1, L_2 or L_3 is a malicious node. Due to the symmetry property of the ANN model, these three faulty situations are equivalent to each other.

For instance, let lieutenant node L_3 to be the faulty node, which is able to send out arbitrary message to confuse other loyal lieutenant nodes. In Figure 5.1a, it shows the first round of message exchange stage where the commander node delivers its order bit "1" to the other three lieutenant nodes. In Figure 5.2a, it shows where the received message bit "1" from the commander node is saved into the message exchange matrix (MEM) of each lieutenant node.



Faulty Processor

Figure 5.1a First round



Faulty Processor

Figure 5.1b Second round

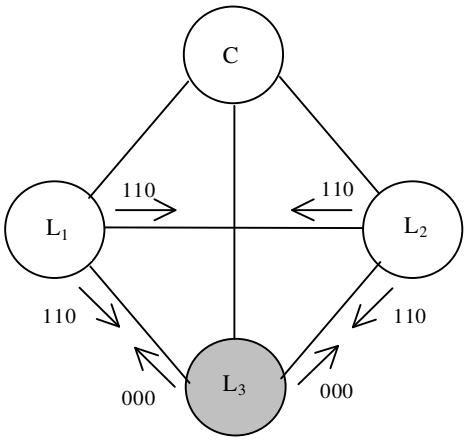


Figure 5.1c Third round

Figure 5.1 The message exchange stage of a 4-processor distributed system

Then in the second round, the loyal lieutenant nodes L_1 and L_2 send the received value in the first round to the other lieutenant nodes. However, for the faulty node L_3, it will send a bit "0" to other lieutenant nodes to confuse the agreement to be made up. The message flow in the second round is shown in Figure 5.1b. Meanwhile Figure 5.2b shows the update of MEMs of each lieutenant node after the second round of message exchange.

L_1		
1		

L_2		
	1	

L_3		
		1

Figure 5.2a MEMs after the first round

L_1		
1		
1		
0		

L_2		
	1	
	1	
	0	

L_3		
		1
		1
		1

Figure 5.2b MEMs after the second round

L_1		
1	1	0
1	1	0
0	0	0

L_2		
1	1	0
1	1	0
0	0	0

L_3		
1	1	1
1	1	1
0	0	1

Figure 5.2c MEMs after the third round

Figure 5.2 MEM at each lieutenant node during the message exchange stage for a 4-processor distributed system

Figure 5.1c shows the delivery of message string with a length of 3 bits from a lieutenant node to the other lieutenant nodes. The exchange of message strings allows the complete formation of MEM at each lieutenant node. Figure 5.2c shows the final update of MEMs. These three MEMs are listed below corresponding to their lieutenant nodes.

$$\text{MEM of } L_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.3}$$

$$\text{MEM of } L_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.4}$$

$$\text{MEM of } L_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

The MEM is now ready to be used as the training data set for the designed artificial neural networks at every node. Among the MEMs of all the lieutenant nodes, it is only necessary to consider the MEMs of the loyal lieutenant nodes. We can in fact neglect the MEMs of faulty lieutenant nodes. As long as Equation (2.3) is complied, the consideration of the MEM at the loyal nodes is sufficient for gaining the BGP solution.

All the loyal lieutenant nodes will use the MEMs formed to train up the weight matrixes and the bias matrixes of the designed ANN model. Once the ANN model is trained up by the BPN learning algorithm, it can be used for future incoming BGP to achieve a consensus among all the loyal nodes. Hence as in this example, the MEMs at the loyal lieutenant nodes L_1 and L_2 is already sufficient to guarantee the achievement of the Byzantine Agreement.

Meanwhile, for the example of 7-processor distributed system, we make the system to tolerate critical case of the BGP malicious situation as in Figure 5.3a. Take node C as the commander node, and L_1, L_2, L_3, L_4, L_5 to L_6 as the lieutenant nodes. For a network with $n = 7$, a maximum of 2 faulty nodes can be tolerated as from Equation (2.3). We assume that L_5 and L_6 are the faulty nodes. Hence L_5 and L_6 may send out arbitrary messages to all the other lieutenant nodes in order to confuse them in making up the final decision called Byzantine Agreement.

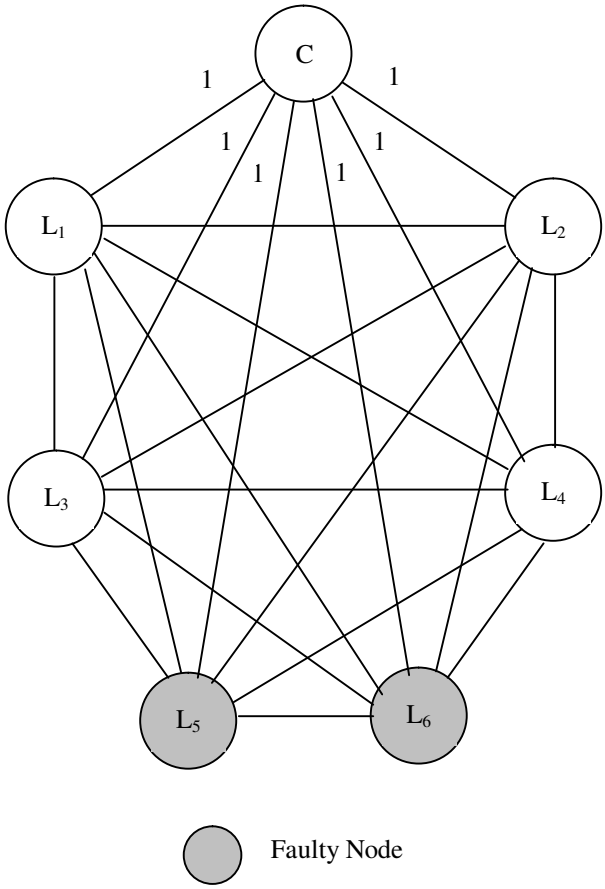


Figure 5.3a First round

Received message at each receiver node					
L_1	L_2	L_3	L_4	L_5	L_6
1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	1	1
1	1	1	1	0	1
1	0	0	0	1	0
0	0	1	0	0	0

Figure 5.3b Second round

Message received by L_1						Local MAJ	Node MAJ
L_1	L_2	L_3	L_4	L_5	L_6		
1	1	1	1	0	1	1	1
1	1	1	1	1	0	1	
1	1	1	1	0	1	1	
1	1	1	1	1	1	1	
1	0	0	0	0	0	0	
0	0	1	0	1	1	DEF	

Message received by L_3						Local MAJ	Node MAJ
L_1	L_2	L_3	L_4	L_5	L_6		
1	1	1	1	0	0	1	1
1	1	1	1	0	1	1	
1	1	1	1	0	1	1	
1	1	1	1	1	1	1	
1	0	0	0	1	0	0	
0	0	1	0	1	0	0	

Message received by L_2						Local MAJ	Node MAJ
L_1	L_2	L_3	L_4	L_5	L_6		
1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	
1	1	1	1	0	0	1	
1	1	1	1	1	0	1	
1	0	0	0	0	0	0	
0	0	1	0	0	0	0	

Message received by L_4						Local MAJ	Node MAJ
L_1	L_2	L_3	L_4	L_5	L_6		
1	1	1	1	0	0	1	1
1	1	1	1	0	1	1	
1	1	1	1	1	1	1	
1	1	1	1	0	0	1	
1	0	0	0	1	1	DEF	
0	0	1	0	0	1	0	

Figure 5.3c Third round of message exchange stage

Figure 5.3 FCN model, message exchange & compromise stage for a 7-node system

There are three rounds of message exchange before the complete formation of MEMs. In the first round, node *C* sends its value of bit "1" to all the lieutenant nodes as in Figure 5.3a. Then in the second round, every lieutenant node sends its value to the other five lieutenant nodes. The messages received by each lieutenant node are shown in Figure 5.3b. Now there are six received messages at each lieutenant node.

In the last round of message exchange message, each lieutenant node sends all the six received messages to the other five lieutenant nodes again. At the end of the third round of message exchange stage, a 6x6 matrix is formed at each lieutenant node. In Figure 5.3c, only the MEMs of loyal lieutenant nodes are shown. This is because the consideration of MEMs at loyal nodes is sufficient in achieving the Byzantine Agreement at the compromise stage later. Those MEMs belonging to the loyal nodes are as below:

$$\text{MEM of } L_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{5.6}$$

$$\text{MEM of } L_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.7}$$

$$\text{MEM of } L_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{5.8}$$

$$\text{MEM of } L_4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

Let us consider the compromise stage for consensus achievement. From each row of the matrix at each node, the local majority (local MAJ) is determined according to the majority function on all the members within the row vector. Then the node majority (node MAJ) is determined from the majority function on all the members within the column vector of local MAJ. Note that DEF stands for DEFAULT value indicating that the number of bit "0" and bit "1" are the same when the majority function is applied.

At the end of the compromise stage, all the node MAJ values for node L_1, L_2, L_3 and L_4 are the same. In addition, a loyal lieutenant node value is the same with the non-faulty commander node value. As a consequence, both the interactive consistency conditions (ICCs) of (C.1) and (C.2) are fulfilled. Lastly, we can see that the ANN based BAP can be applied to solve the BGP.

5.3 BIT COMPOSITION OF MESSAGE EXCHANGE MATRIX

Message exchange matrix (MEM) is an (n-1)*(n-1) matrix formed after the message exchange stage. This MEM is used as the training data set during the training stage of the ANN based BAP. Meanwhile when the neural network is trained up, MEM is used as the testing data set in the application stage.

For an n-processor distributed network, a maximum of m faulty processors can be tolerated as in Equation (2.3). Assume the commander processor to be loyal, then there are m faulty lieutenant processors. Hence as we take the distributed network as a graph of n nodes, we have one commander node and (n-1) lieutenant nodes. Among the (n-1) lieutenant nodes, m of them is faulty nodes. These (n-1)

lieutenant nodes are represented by (n-1) columns of the MEM, where m of the columns representing the faulty nodes.

The column of the MEM represents the receiver nodes, and the row of the MEM represents the sender nodes. Hence we have m rows of faulty message senders and m columns of faulty message receivers. Due to the symmetrical property of the ANN architecture, the sequence number of the faulty rows and faulty columns are independent from the achievement of Byzantine Agreement. As long as the number of faulty nodes is at a maximum of m nodes, the Byzantine Generals Problem of the unsigned message can be solved with the ANN based BAP.

As the sequence of rows sent by the lieutenant nodes is independent from the neural network training, we can gather the rows sent by the faulty nodes together. For example, let us take the cases of n = 4 and 7. For n = 4, the maximum value of m is 1. Therefore, either one of L_1, L_2, or L_3 can be faulty. However, these three faulty cases are equivalent to each other. Hence we can always make some denotation manipulations by assuming L_1 to be the faulty node.

For case n = 7, let L_1 and L_2 be the faulty nodes out of a list of lieutenant nodes (L_1, L_2, L_3, L_4, L_5 and L_6). The case where L_1 and L_2 are the faulty nodes is the same with the case of the combination of any two faulty lieutenant nodes. In general, there will be one commander node C and (n-1) lieutenant nodes (L_i : L_1, L_2, L_3, …, L_{n-1}) in an n-processor distributed system. Under the critical malicious circumstances, there are $m = \lfloor (n\text{-}1)/3 \rfloor$ faulty nodes as from Equation (2.3).

To make the analysis easier, we arrange the MEM so that the faulty rows are always at the top of the MEM. In other words, the first m rows of the MEM are rows of messages sent by the faulty lieutenant nodes. In addition, the first m columns from the left side of the MEM are arranged to be faulty columns as well. Therefore, we have a MEM as in Figure 5.4 for analysis.

From Figure 5.4, we notice that MEM contains both loyal and faulty bits. The total bits of MEM, number of loyal bits and number of faulty bits are given below.

$$\text{Total bits of MEM} = (n\text{-}1)^* (n\text{-}1)$$

$$= (n\text{-}1)^2 \qquad (5.10)$$

Total loyal bits of MEM = $(n\text{-}m\text{-}1)^* (n\text{-}m\text{-}1)$

$$= (n\text{-}m\text{-}1)^2 \quad (5.11)$$

Total faulty bits of MEM = $m^*m + m^*(n\text{-}m\text{-}1) + (n\text{-}m\text{-}1)^*m$

$$= m^*(2n\text{-}m\text{-}2) \quad (5.12)$$

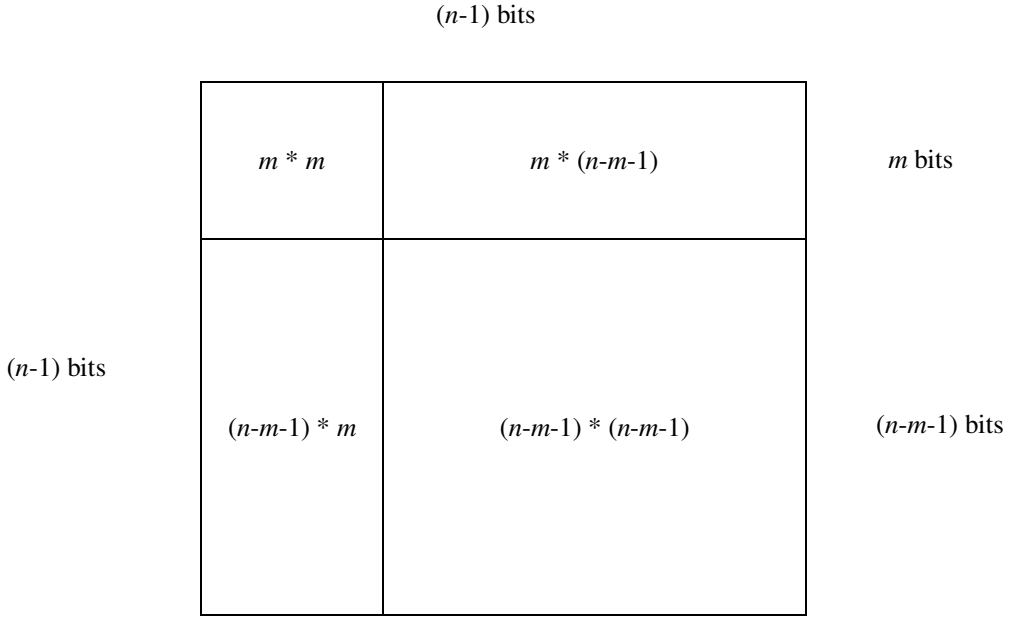


Figure 5.4 Message exchange matrix for an *n*-processor distributed system

5.4 MATHEMATICAL ANALYSIS OF MESSAGE EXCHANGE MATRIX

To analyze the message exchange matrix (MEM), we have some denotations as shown below:

n = number of nodes in the network

m = number of faulty nodes in the network

npr = total number of bits in the MEM = $(n\text{-}1)^2$

nloy = number of bits of the loyal lieutenant nodes

nfau = number of bits of the faulty lieutenant nodes

v_{loy} = number of row vectors sent by loyal lieutenant nodes

v_{fau} = number of row vectors sent by faulty lieutenant nodes

v_{npr} = total number of row vectors

Assuming the commander node is loyal, we have m = number of faulty lieutenant nodes. In addition, we can have some parameter relationships as shown below:

$n = \{ 4, 5, 6, 7, \ldots \}$

$m = \{ 0 \leq m \leq \text{floor}((n\text{-}1)/3) \}$

where floor(x) is a function to round the value of x to the nearest integer smaller or equal to x.

$npr = (n\text{-}1)^2$

$nloy = (n\text{-}m\text{-}1)^* (n\text{-}m\text{-}1) = (n\text{-}m\text{-}1)^2$

$nfau = m^*m + m^*(n\text{-}m\text{-}1) + (n\text{-}m\text{-}1)^*m = m^*(2n\text{-}m\text{-}2)$

$npr = nloy + nfau$

$v_{npr} = v_{loy} + v_{fau}$

For an n-node distributed network, the tolerance of the number of faulty nodes, m, ranges from 0 to floor((n-1)/3). To analyze the effect of m on the MEM, we set the value of m to the worst-case situation where m = floor((n-1)/3). For every critical value of m, there exist three corresponding values of n. Let $m = u$ where $u = \{ 1, 2, 3, 4, \ldots \}$, we can have three cases of n as shown below:

Case 1: $n = 3u + 1$ (C1)

Case 2: $n = 3u + 2$ (C2)

Case 3: $n = 3u + 3$ (C3)

For all the three cases above, we calculate some ratios to help the MEM analysis. These ratios are (*nloy*/*npr*), (*nfau*/*npr*), (v_{loy}/v_{npr}) and (v_{fau}/v_{npr}). The results of the ratio computation for every case are listed in Table 5.1. From the table, we can notice the trend of the loyal bit composition of a MEM as the size of the network is growing up.

Table 5.1 Various ratios for MEM analysis on the worst case of *m*

Case	C1 : $n = 3u + 1$	C2 : $n = 3u + 2$	C3 : $n = 3u + 3$
$\left(\frac{nloy}{npr}\right)$	$\frac{4}{9}$	$\left(\frac{2u+1}{3u+1}\right)^2$	$\left(\frac{2u+2}{3u+2}\right)^2$
$\left(\frac{nfau}{npr}\right)$	$\frac{5}{9}$	$\frac{5u^2+2u}{9u^2+6u+1}$	$\frac{5u^2+4u}{9u^2+12u+4}$
$\left(\frac{v_{loy}}{v_{npr}}\right)$	$\frac{2}{3}$	$\frac{2u+1}{3u+1}$	$\frac{2u+2}{3u+2}$
$\left(\frac{v_{fau}}{v_{npr}}\right)$	$\frac{1}{3}$	$\frac{u}{3u+1}$	$\frac{u}{3u+2}$

By comparing case 1, case 2 and case 3, we have some interesting discoveries. From case 1 to case 3, the ratio of (*nloy*/*npr*) and (v_{loy}/v_{npr}) increases, while the ratio of (*nfau*/*npr*) and (v_{fau}/v_{npr}) decreases. Hence the density of the loyal bit in a MEM increases from case 1 to case 3. This will allow the Byzantine Agreement to be achieved easier and faster from case 1 to case 3. These relationships are expressed mathematically in Equations (5.13-5.16).

$$\left(\frac{nloy}{npr}\right)_{C1} < \left(\frac{nloy}{npr}\right)_{C2} < \left(\frac{nloy}{npr}\right)_{C3} \tag{5.13}$$

$$\left(\frac{v_{loy}}{v_{npr}}\right)_{C1} < \left(\frac{v_{loy}}{v_{npr}}\right)_{C2} < \left(\frac{v_{loy}}{v_{npr}}\right)_{C3} \tag{5.14}$$

$$\left(\frac{nfau}{npr}\right)_{C1} > \left(\frac{nfau}{npr}\right)_{C2} > \left(\frac{nfau}{npr}\right)_{C3} \tag{5.15}$$

$$\left(\frac{v_{fau}}{v_{npr}}\right)_{C1} > \left(\frac{v_{fau}}{v_{npr}}\right)_{C2} > \left(\frac{v_{fau}}{v_{npr}}\right)_{C3} \tag{5.16}$$

5.5 EFFECT OF INCREASING NETWORK SIZE

Using the mathematical analysis from Section 5.4, we can interpret the effects of increasing the size of the distributed network. As the size of the network, *n*, grows up, case 2 and case 3 will converge to case 1. Furthermore, the required mean epoch to train the artificial neural network decreases as the number of node increases as in Figure 4.19. This phenomenon is explained as below.

As the number of node *n* increases, the number of bits in the message exchange matrix, which is $(n\text{-}1)^2$, increases in the order of n^2. The pattern of bits in the MEM is classified into its corresponding classes by using ANN. A trained ANN does its pattern classification task by feeding the row vectors of the MEM into its input neurons in the input layer row by row.

In a MEM, all the loyal bit positions will share the common value. Therefore, there is always a minimum of $(n\text{-}m\text{-}1)^2$ bits out of a total of $(n\text{-}1)^2$ sharing the same bit value. Meanwhile, for faulty bit positions, they are all having randomly generated arguments. If all the loyal bit locations are bit "1", then the faulty bit locations can be either bit "0" or bit "1".

We can have three significant situations in the composition of faulty bit locations. There are two extreme cases and one average case. The extreme cases are the worst-case situation and the best-case situation. Worst-case situation means all the faulty bit positions carry the values different from the loyal bit value at the loyal bit positions. Meanwhile, best-case situation means all the faulty bit positions carry the values equivalent to the loyal bit value. Average-case situation means half of the faulty bit positions carry the values equivalent to the loyal bit value.

Instead of the factor of random initialisation of weights and biases, which causes the epoch of ANN training to fluctuate, the bit composition of the MEM as an input training matrix is another factor as well. The worst-case situation causes the ANN training to have maximum number of epoch. The best-case situation requires minimum number of epoch for ANN training. On the other hand, the average-case situation asks for an average epoch.

For small value of *n*, the extreme cases are most likely to occur, and hence the difference between the minimum and maximum epoch for FCN-*n* with small *n* tends to be wider than FCN-*n* with large *n*. For large value of *n*, the bit composition of the faulty bit positions tends to be the average case, and hence it is most likely to ask for an average epoch. These phenomena can be explained by using the games as elaborated below.

On the analogy of the game of throwing a coin where our task is to guess its value of either FACE or HEAD, and the game of throwing a dice, we have non-evenly distributed samples for small number of trials. However, as the number of nodes, *n*, becomes sufficiently large, the sample will be evenly distributed for each value according to its probability of occurrence. For instance, coin throwing will have the probability of 1/2 for both FACE and HEAD. Throwing a dice will have the probability of 1/6 for each of the values of 1, 2, 3, 4, 5 and 6.

Hence as the distributed network size, *n*, increases, the bit "0" and bit "1" in the faulty bit position will be evenly distributed with the probability of occurrence of 0.5 for each binary bit. At the same time, the loyal bit positions always carry the same bit value of either bit "0" or bit "1". Hence as *n* increases, the number of bit value belongs to the loyal bit value will tend to become an average value. In other words, it means the sample is getting away from the worst-case situation and the best-case situation towards the average-case situation.

If *n* is big enough, and let *nloybit* as the number of bit carrying the loyal bit value, we have a ratio as follows:

$$(\frac{nloybit}{npr}) = \frac{1}{npr}[nloy + floor(\frac{nfau}{2})] \tag{5.17}$$

As n is sufficiently large, case 2 and case 3 will converge to case 1. Hence the consideration of case 1 where $n = 3m+1$ will be sufficient to see the effect of increasing distributed network size on the bit composition of MEM. We have:

$$\left(\frac{nloybit}{npr}\right)_{C1} = \left(\frac{nloy}{npr}\right)_{C1} + \frac{1}{npr} floor\left(\frac{nfau}{2}\right)$$

$$= \frac{4}{9} + \frac{1}{npr} floor\left(\frac{nfau}{2}\right) \tag{5.18}$$

Therefore as the value of n increases, it is easier to recognize the pattern of the bit composition of message exchange matrix. Indirectly it means that it is easier to train the neural network. Hence, we can finally deduce that the required mean epoch to train the neural network reduces while the distributed network is growing bigger.

To test the mathematical analysis on effects of increasing network size, some experiments are conducted using the standard BPNN without bias training algorithm. Table 5.2 shows the epoch needed to train up the ANN using standard BPNN without bias. Notice the relationships between the n, m and percentage of loyal nodes (%loyal node). For every step of m, there are three corresponding values of n. Compare each n with the rule $mod(n,3)$ = 0, 1 or 2, where $mod(n,3)$ is the remainder of n/3.

Within each class of $mod(n,3)$, it can be found out from Table 5.2 that the epoch needed reduces for every increasing step of m. For the class of $mod(n,3)$ = 1, it is n = 4, 7 and 10 as shown in Figure 5.5. For the class of $mod(n,3)$ = 2, it is n = 5, 8 and 11 as shown in Figure 5.6. For the class of $mod(n,3)$ = 0 it is n = 6, 9 and 12 as in Figure 5.7.

Table 5.2 Epoch needed for standard BPNN training of *n*-processor distributed system

n	*m*	%loyal node	Network Complexity	Epoch		
				Minimum	Maximum	Mean
4	1	75.0	18	2126	3528	3236.2
5	1	80.0	28	1205	3478	2210.4
6	1	83.3	32	368	3185	1596.6
7	2	71.4	45	704	1157	1032.6
8	2	75.0	50	522	1186	789.8
9	2	77.8	64	285	1054	678.2
10	3	70.0	72	178	630	346.6
11	3	72.7	91	160	486	286.0
12	3	75.0	98	78	264	126.0

On the other hand, for m = 1, 2 and 3, it can be noticed that the epoch needed reduces from n = 4 to n = 6, from n = 7 to n = 9, and from n = 10 to n = 12 respectively. For the case of m = 1, it is shown in Figure 5.8 for n = 4, 5 and 6. For the case of m = 2, it is shown in Figure 5.9 for n = 7, 8 and 9. For the case of m = 3, it is shown in Figure 5.10 for n = 10, 11 and 12. All of these experiments are run to obtain the MSE error limit of 0.001 over a Pentium PC 266MHz.

In general, the required epoch to train a neural network model for ANN based BAP application depends on number of nodes in a distributed network (n), number of faulty nodes (m), neural architecture, network complexity (or interconnection complexity), types of BPNN learning algorithm and required MSE error limit.

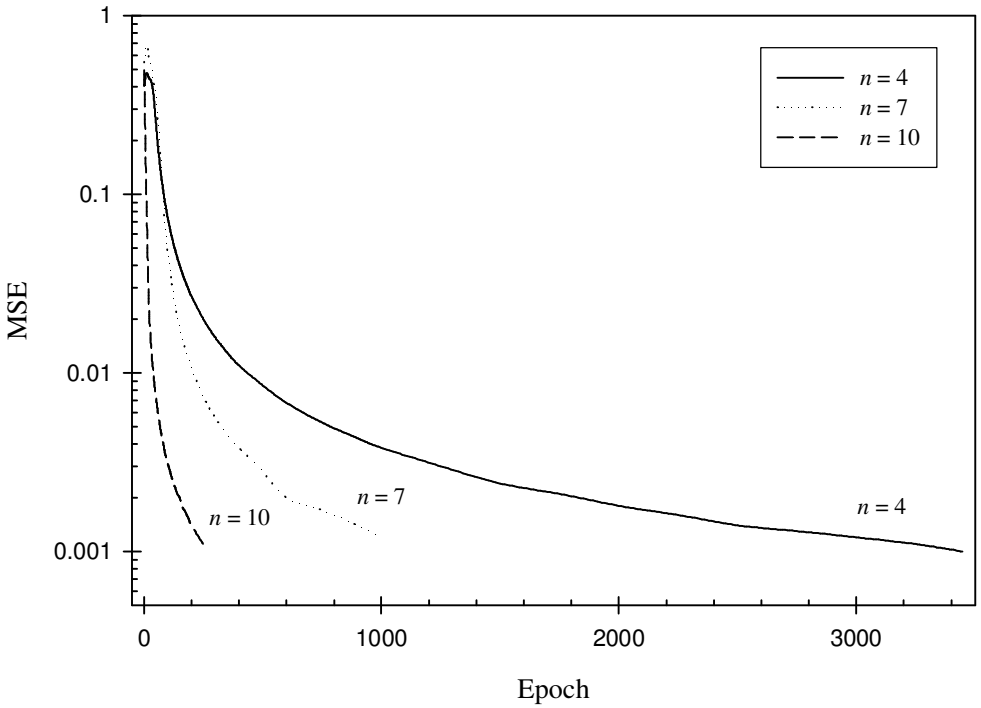


Figure 5.5 Required epoch for *n*-node network of *n* = 4, 7 and 10 (case *mod(n,3)* = 1)

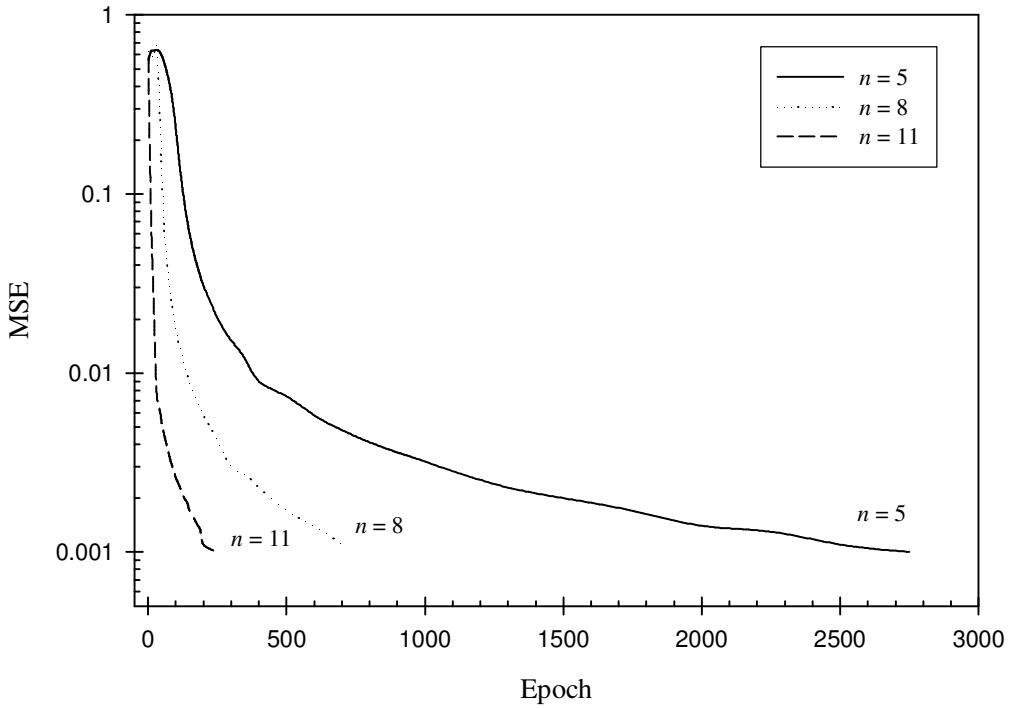


Figure 5.6 Required epoch for *n*-node network of *n* = 5, 8 and 11 (case *mod(n,3)* = 2)

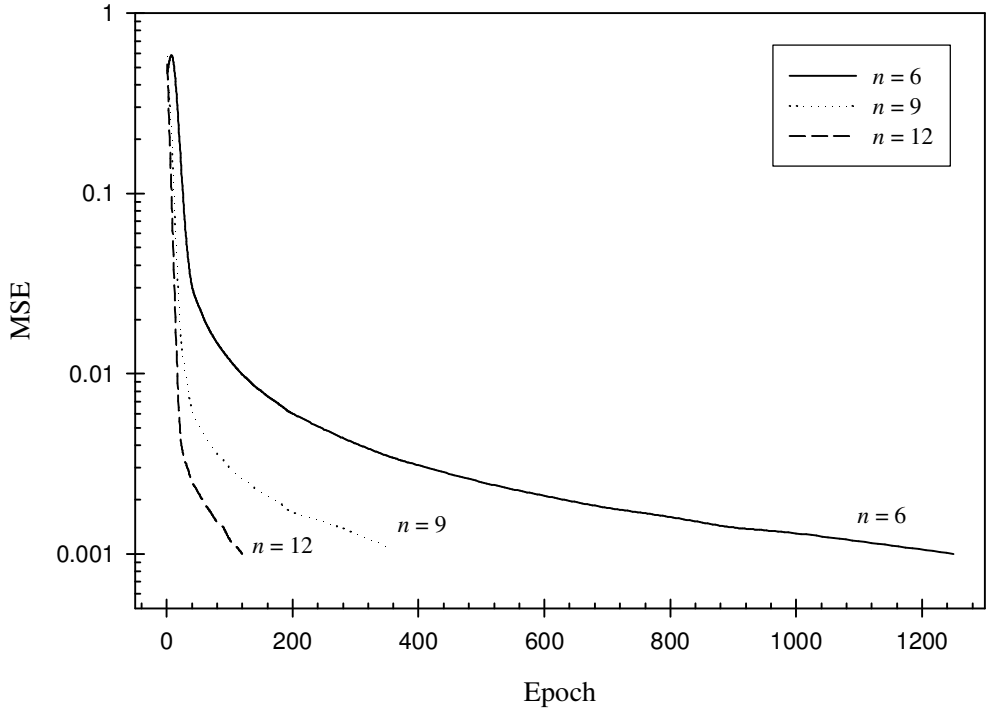


Figure 5.7 Required epoch for *n*-node network of $n = 6$, 9 and 12 (case *mod(n,3)* = 0)

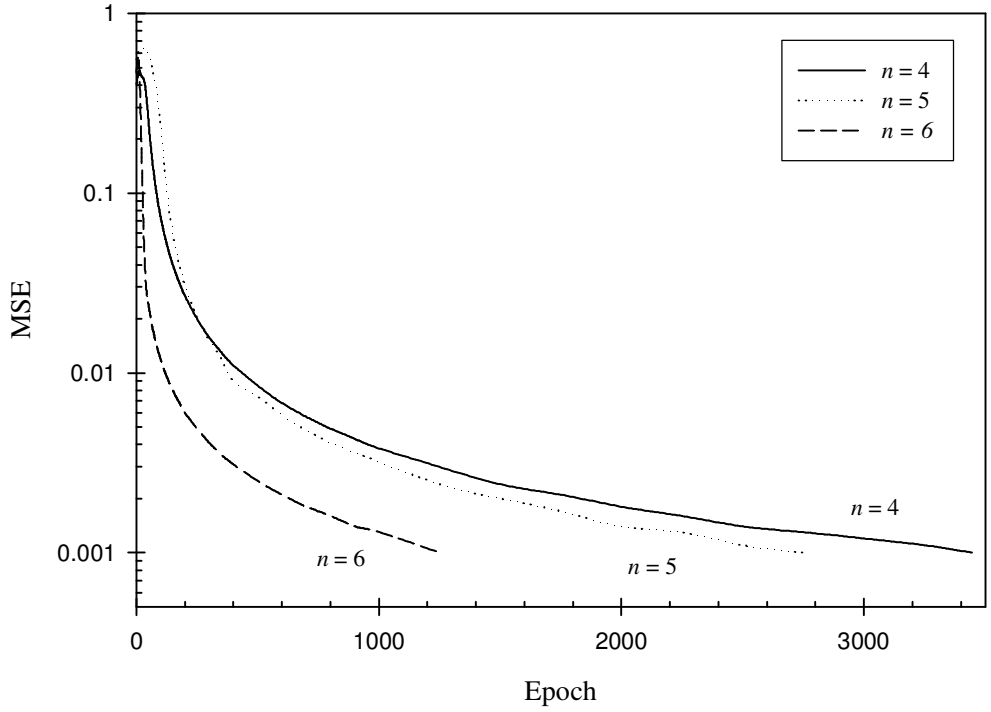


Figure 5.8 Required epoch for *n*-node network of $n = 4$, 5 and 6 (case $m = 1$)

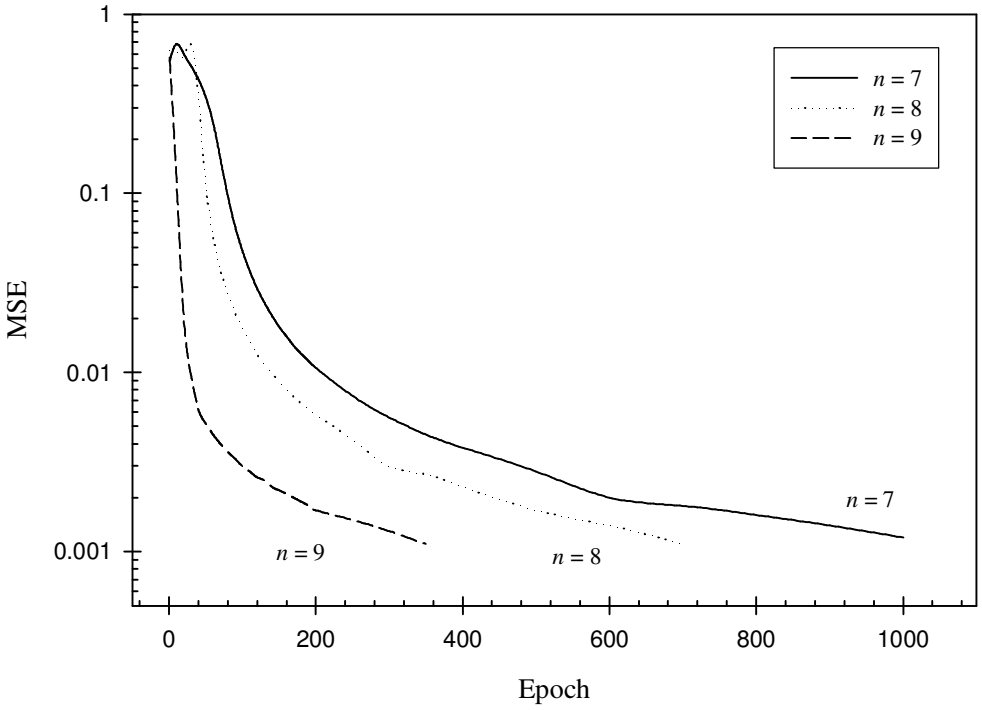


Figure 5.9 Required epoch for n-node network of n = 7, 8 and 9 (case m = 2)

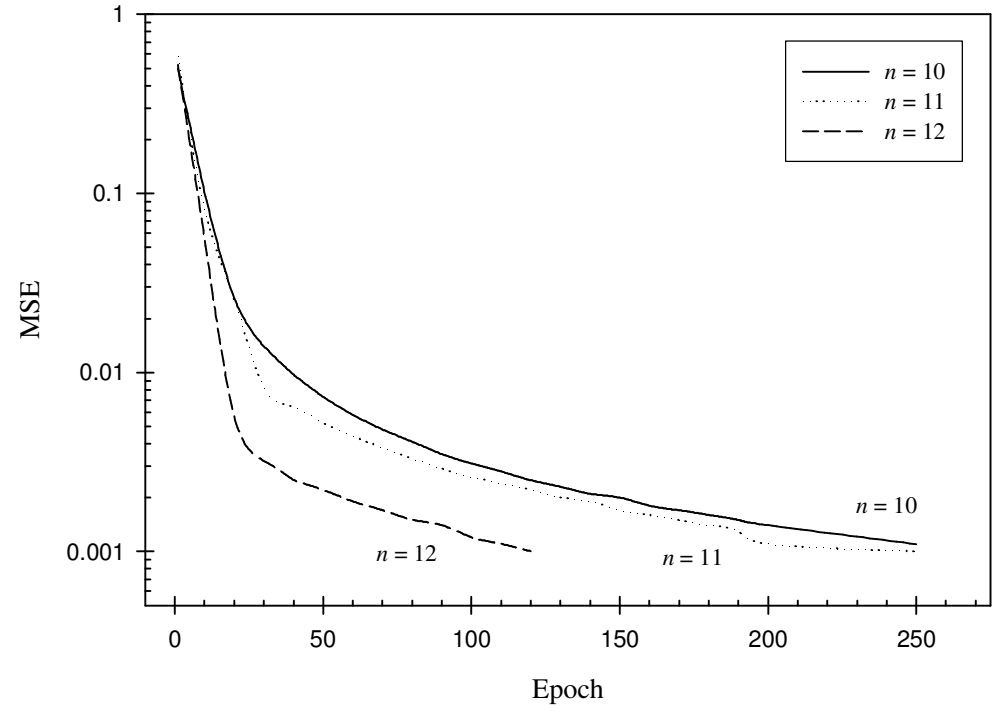


Figure 5.10 Required epoch for n-node network of n = 10, 11 and 12 (case m = 3)

5.6 CONCLUSIONS

In this chapter, the message exchange message matrix (MEM) is analyzed. The functions of MEM as both the training set in the training stage and the testing set in the application stage are discussed. In Section 5.2, the formation of MEM in the message exchange stage is presented for the general case of n-processor distributed system and specific cases of $n = 4$ and $n = 7$.

In Section 5.3, the bit composition of MEM is discussed from the angle of the loyal bits and faulty bits. Then the locations of loyal bits and faulty bits in the MEM matrix are shown together with the calculation. In Section 5.4, the MEM is analyzed mathematically upon its bit compositions. This computational analysis is based on the consideration of the existence of three faulty cases for every value of m, which is the number of faulty nodes in an n-node network.

After that, the deduction from the mathematically analyzed MEM on the effects of increasing network size is discussed. It is found that the loyal bit density of MEM increases while the size of the distributed network gets larger. The increment of loyal bit density allows less mean epoch to train the neural network. Therefore ANN based BAP has an advantage over the traditional BAP as the network size becomes bigger. ANN based BAP will achieve Byzantine Agreement faster than the traditional BAP for $n \geq 10$ as shown in Figure 4.66.

From the mathematical analysis and experimental results, it is discovered that for fixed number of faulty nodes, the required epoch for BPNN training reduces, as the network becomes bigger. Furthermore, if the networks are categorized according to *mod(n,3)*, which is the remainder of $n/3$, there is another reducing pattern of the required epoch. For networks having the same value of *mod(n,3)*, the required epoch for BPNN training decreases while the size of the network is growing larger.

CHAPTER 6: FAULTY NODE DETECTION

6.1 INTRODUCTION

In all of the previous chapters, the discussions are mainly encircling the topics of Byzantine Generals Problem (BGP) in the dependable distributed systems, and the application of artificial neural network (ANN) for solving the BGP by ANN based BAP. In other words, it is all about the achievement of common consensus called Byzantine Agreement.

At the end of the traditional BAP and ANN based BAP, each loyal node solves the unanimity problem of BGP by achieving the same agreement. Each loyal node knows that all the loyal nodes within the same distributed network hold the same Byzantine Agreement, but each loyal node is ignorant of the identity of the loyal nodes and faulty nodes. To enable each loyal node to be aware of the faulty status of other nodes, the message exchange matrix (MEM) has to be investigated again. In this chapter, the topic on the detection of faulty node will be discussed (Lee & Ewe, 2003b).

6.2 HIDDEN INFORMATION IN THE MESSAGE EXCHANGE MATRIX

The functions of MEM is not limited to being the training set in the message exchange stage and the testing set in the application stage of the ANN based BAP. It is also useful in helping the loyal node to be aware of the faulty status of other nodes in the same distributed network. In Section 5.2, the formation of MEM is presented. In Section 5.3, the bit composition of MEM classified according to the loyal bit and faulty bit is discussed.

In fact, instead of the apparent information of bit contents, there is another hidden information within the MEM matrix. The specific position of each bit in the MEM contains the implicit information about the sender and receiver of that particular bit. Instead of considering the faulty bit and loyal bit within a MEM matrix as in Figure 5.4, this time the focus is given on the relationship of every single member of the MEM and its specific coordinate.

We start our consideration from the simplest case of BGP, which is the 4-processor distributed system. Again, let the 4-processor distributed system to be a graph of four nodes as in Figure 4.14. The 4-node graph, which is a FCN-4, is assumed to have one commander node C and three lieutenant nodes L_1, L_2 and L_3. Let C, L_1 and L_2 to be the loyal nodes and L_3 to be the faulty node. Another assumption is the links of the FCN-4 are free of any arbitrary fault.

The simulated FCN-4 will generate a MEM matrix having the dimension of 3x3 at each lieutenant node. The way the MEM is formed during the message exchange stage is illustrated in Figure 5.1. Let the coordination of the MEM to be in terms of row i and column j, and each bit position is labelled as P_{ij}. The row i represents the sender node, and the column j represents the receiver node. A position-coordinated MEM at each lieutenant node is obtained as in Figure 6.1.

Column j

Row i			
	P_{11}	P_{12}	P_{13}
	P_{21}	P_{22}	P_{23}
	P_{31}	P_{32}	P_{33}

Figure 6.1 MEM of FCN-4 with its bit positions labelled as P_{ij}

The MEM coordination explained in this paragraph is from the perspective of lieutenant node L_1. For coordinate P_{11}, it contains the bit message sent by the commander node C in the first round of message exchange. Positions P_{21} and P_{31} contain the bit messages from the lieutenant node L_2 and L_3 respectively during the

second round of message exchange. The bit messages at positions P_{21} and P_{31} are in fact sent from the commander node to the lieutenant nodes L_2 and L_3 during first round of communication. Both L_2 and L_3 receive the bit message from C and send it to all the lieutenant nodes in the second round of communication. At the end of second round of message exchange, each lieutenant node holds three bit messages. These three bit messages are delivered to all the other lieutenant nodes in the third round of message exchange. At the lieutenant node L_1, the bit messages sent by the L_2 are put at the positions of P_{12}, P_{22} and P_{32}. On the other hand, the bit messages sent by L_3 are put at the positions of P_{13}, P_{23} and P_{33}.

From the perspective of lieutenant node L_2, coordinate P_{22} contains the bit message sent by the commander node C in the first round of message exchange. Positions P_{12} and P_{32} contain the bit messages from the lieutenant node L_1 and L_3 respectively during the second round of message exchange. The bit messages at positions P_{12} and P_{32} are in fact sent from the commander node to the lieutenant nodes L_1 and L_3 during first round of communication. Both L_1 and L_3 receive the bit message from C and send it to all the lieutenant nodes in the second round of communication. During the third round of communication, the bit messages sent by the L_1 are put at the positions of P_{11}, P_{12} and P_{13}. Concurrently, the bit messages sent by L_3 are put at the positions of P_{13}, P_{23} and P_{33}.

From the perspective of lieutenant node L_3, coordinate P_{33} contains the bit message sent by the commander node C in the first round of message exchange. Positions P_{31} and P_{32} contain the bit messages from the lieutenant node L_1 and L_2 respectively during the second round of message exchange. The bit messages at positions P_{31} and P_{32} are in fact sent from the commander node to the lieutenant nodes L_1 and L_2 during first round of communication. Both L_1 and L_2 receive the bit message from C and send it to all the lieutenant nodes in the second round of communication. During the third round of communication, the bit messages sent by the L_1 are put at the positions of P_{11}, P_{12} and P_{13}. Concurrently, the bit messages sent by L_2 are put at the positions of P_{21}, P_{22} and P_{23}.

For a general case of n-node network, we will have a MEM with the dimensions of $(n\text{-}1)*(n\text{-}1)$ at each lieutenant node. For the MEM at lieutenant node

L_g, where $1 \le g \le i$ and $1 \le g \le j$, the source message from the commander node C in the first round of communication will be saved in coordinate P_{gg}. In the second round of message exchange, the message received from lieutenant node L_h, where $1 \le h \le i$, $1 \le h \le j$ and $h \ne g$, will be saved in position P_{hg}. At the end of second round, the column vector g of MEM at L_g shall have been full. In the third round of communication, every lieutenant node sends (n-1) messages to all the other lieutenant nodes. The string messages from lieutenant node L_h will be kept at positions P_{ih} of the MEM in node L_g. At the end of the third round of message exchange, the MEM in node L_g is then fully constructed, as well as the MEM in the other lieutenant nodes.

Therefore, we can deduce the identity of the sending node and receiving node during the rounds of message exchange from the denotation of the coordinate P_{ij} of a MEM in lieutenant node L_g. Since there are three rounds of message exchange, the types of coordinate P_{ij} can be classified into three groups as in Table 6.1. The first group is from the first round of communication. The second group is a mixture of second and third rounds of communication. The third group is from the last round of message exchange.

Table 6.1 Types of coordinate P_{ij} of MEM in node L_g

Types of P_{ij}	Interpretation
$i = j = g$	The message is from commander node C to lieutenant node L_g.
$i \ne j$ and $j = g$ or $i = j \ne g$	The message is from lieutenant node L_i to lieutenant node L_g, claiming that this is the message it received from commander node C in the previous round of communication.
$i \ne j$ and $j \ne g$	The message is from lieutenant node L_j to lieutenant node L_g, claiming that this is the message it received from lieutenant node L_i in the previous round of communication.

6.3 FAULTY NODE DETECTION

Applying the sender and receiver information together with the message content of each coordinate in the MEM, it is possible to detect the faulty node within a malicious distributed network. The MEM ability on faulty node detection is possible by comparing its row vector majority and column vector majority with the MEM matrix majority.

The row vector majority of MEM is interpreted as *local majority* in the examples given in Figure 5.3. For every MEM in a node, there are (n-1) row vectors. Hence we have (n-1) local majorities corresponding to (n-1) lieutenant nodes. The local majority is computed by applying the majority function upon set of messages belonging to a row vector.

For the example of FCN-4 given in Figure 5.2, the results of the majority function on the row vectors of MEM are illustrated in Figure 6.2. Since lieutenant node L_3 is a faulty node and consideration of loyal nodes is sufficient for fulfilling the interactive consistency conditions (ICCs) of (C.1) and (C.2), Figure 6.2 shows only the local majorities of the MEMs in loyal lieutenant nodes L_1 and L_2. For both of the loyal lieutenant nodes L_1 and L_2, the first row and the second row of their MEMs give a local majority of bit "1". Their third rows give a local majority of bit "0". First row represents the message from L_1, second row is message from L_2, and third row is message from L_3.

MEM in L_1			Local MAJ	Node MAJ
1	1	0	1	1
1	1	0	1	
0	0	0	0	

MEM in L_2			Local MAJ	Node MAJ
1	1	0	1	1
1	1	0	1	
0	0	0	0	

- MAJ - majority

Figure 6.2 Local majority and node majority of the MEMs of FCN-4 in Figure 5.2

Meanwhile the MEM matrix majority is interpreted as the *node majority*. Node majority of a MEM is computed from the majority function on the local majority of the row vectors. It is a decision made by a lieutenant node on the message it received from the commander node. From the example of FCN-4, the node majority for both lieutenant nodes L_1 and L_2 is bit "1". This bit "1" is in fact the Byzantine Agreement among all the loyal nodes in a malicious network. For more details, refer to the illustration in Table 2.2 and the explanation in Section 2.7.

From Section 6.2 and Table 6.1, we know that the number of row vector of a MEM indicates the sender of the message stored in a row vector. The receiver of the message is then the number of node in which the MEM resides. Therefore, by taking the majority function on the row vectors of the MEM, we can somehow deduce the faulty status of other lieutenant nodes from the comparison of local majority and node majority. This is because node majority is the Byzantine Agreement among all the loyal nodes in a malicious network. The deviation of any local majority from node majority indicates the possibility of a corresponding lieutenant node to be a traitorous node.

As an example, refer to the FCN-4 in Figure 6.2 again, the local majority of the first row and second row agree with the node majority for both the lieutenant nodes L_1 and L_2. This hints that both L_1 and L_2 are loyal lieutenant nodes. However, although the local majority of the third row vector has a value of bit "0", which is differed from the node majority, it is not yet sufficient to confirm that L_3 is malicious. The difference of row vector majority from node majority can only indicate that the faulty status of lieutenant node L_3 is suspicious. The malevolence of L_3 has to be double-checked by another parameter called column vector majority.

Column vector majority is the value computed by applying the majority function over the column vector of the MEM. Therefore, same like the row vector majority, there are altogether a total of (n-1) column vector majority for an n-node network. Why column vector majority is necessary in determining the faulty status of a particular lieutenant node? This can be explained by using the example of FCN-4 again, where we will show that both the cases of faulty lieutenant node and faulty commander node can have the same set of row vector majority.

Instead of the lieutenant node L_3, this time we let the commander node to be malevolent. In other words, it means instead of the fault situation as in Figure 2.7, the fault situation in Figure 2.8 is adopted. Commander node C becomes a faulty node, whereas lieutenant nodes L_1, L_2 and L_3 become the loyal nodes.

Therefore, the rounds of message exchange in Figure 5.1 have to be reviewed as well as the constructed MEM in each lieutenant node in Figure 5.2. For the FCN-4 with a faulty commander node C, the corresponding message exchange stage and MEM at each lieutenant node are in Figure 6.4 and 6.3 respectively. Concurrently the local majority and the node majority of MEM in each loyal lieutenant node of the FCN-4 have to be revised into Figure 6.5 as compared to Figure 6.2.

L_1		
1		

L_2		
	1	

L_3		
		0

Figure 6.3a MEMs after the first round

L_1		
1		
1		
0		

L_2		
	1	
	1	
	0	

L_3		
		1
		1
		0

Figure 6.3b MEMs after the second round

L_1		
1	1	1
1	1	1
0	0	0

L_2		
1	1	1
1	1	1
0	0	0

L_3		
1	1	1
1	1	1
0	0	0

Figure 6.3c MEMs after the third round

Figure 6.3 MEM at each lieutenant node during the message exchange stage of FCN-4 with a faulty commander node

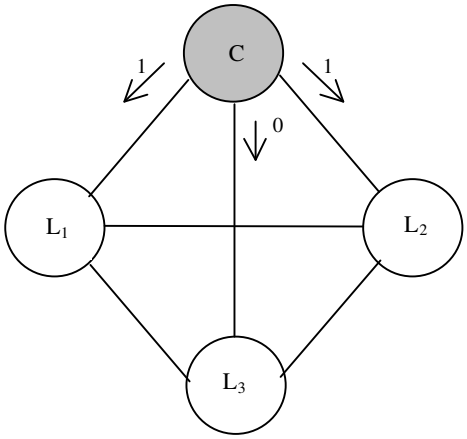


Figure 6.4a First round

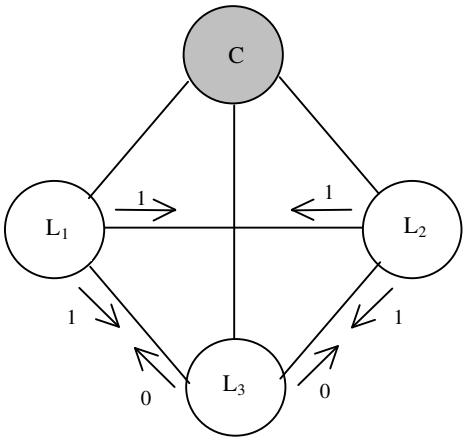


Figure 6.4b Second round

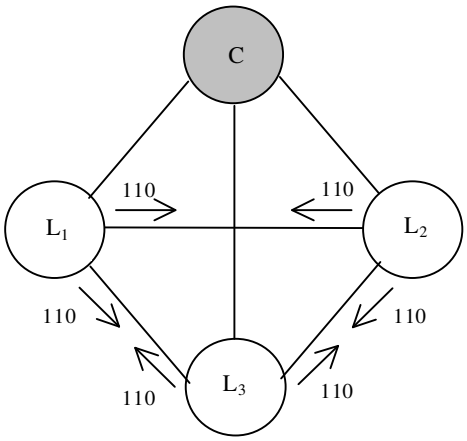


Figure 6.4c Third round

Figure 6.4 The message exchange stage of FCN-4 with a faulty commander node

As from Figure 6.5 for the case of FCN-4 with a faulty commander node *C*, the node majority of all the loyal lieutenant nodes L_1, L_2 and L_3 are same. It means that the ICCs of (C.1) and (C.2) are complied and the Byzantine Agreement is achieved among all the loyal nodes regardless of the malevolence of commander node *C*.

	MEM in L_1			Local MAJ	Node MAJ
	1	1	1	1	1
	1	1	1	1	
	0	0	0	0	
CM	1	1	1		

	MEM in L_2			Local MAJ	Node MAJ
	1	1	1	1	1
	1	1	1	1	
	0	0	0	0	
CM	1	1	1		

	MEM in L_3			Local MAJ	Node MAJ
	1	1	1	1	1
	1	1	1	1	
	0	0	0	0	
CM	1	1	1		

- MAJ - majority
- CM - column vector majority

Figure 6.5 Local majority, node majority and column vector majority of the MEMs of FCN-4 with a faulty commander node

Comparing the node majority and the local majority of each row vector for each MEM in lieutenant nodes L_1 and L_2 as in Figures 6.2 and 6.5, it is found that all the row vectors are sharing the same local majority. This indicates that both of the lieutenant nodes L_1 and L_2 are loyal, whereas the faulty status of the lieutenant node L_3 is suspicious.

Even though Figure 2.7 refers to the FCN-4 with a faulty lieutenant node L_3, and Figure 2.8 refers to the FCN-4 with a faulty commander node C, we cannot differentiate these two fault situations merely based on the node majority and local majority (or row vector majority). As a consequence, we cannot say affirmatively that whether commander node C or lieutenant node L_3 is malicious. We only know that there exists a faulty node within the FCN-4 distributed system. To investigate further, it requires the column vector majority to check the faulty status of C and L_3. Therefore, Figure 6.2 is revised to become Figure 6.6.

	MEM in L_1			Local MAJ	Node MAJ
	1	1	0	1	1
	1	1	0	1	
	0	0	0	0	
CM	1	1	0		

	MEM in L_2			Local MAJ	Node MAJ
	1	1	0	1	1
	1	1	0	1	
	0	0	0	0	
CM	1	1	0		

- MAJ - majority
- CM - column vector majority

Figure 6.6 Local majority, node majority and column vector majority of the MEMs of FCN-4 with a faulty lieutenant node

Comparing Figures 6.5 and 6.6, it can be found that the column vector majority for both fault situations of FCN-4 has a difference. For the first and the second columns, both FCN-4 give a value of bit "1" as the column vector majority. This value of bit "1" agrees with the Byzantine Agreement as in the node majority of bit "1". It implies again that both lieutenant nodes L_1 and L_2 are loyal nodes.

Nevertheless, in the third column of the MEMs in lieutenant nodes L_1 and L_2, FCN-4 with a faulty lieutenant node gives a column vector majority of bit "0". On the contrary, FCN-4 with a faulty commander node gives a column vector majority of bit "1" at its third column of MEMs in nodes L_1 and L_2. This hints that lieutenant node L_3 is a loyal node as well. The faulty node in Figure 6.5 is diagnosed to be the

commander node *C*. Meanwhile the faulty node in Figure 6.6 is found to be the lieutenant node L_3.

As a result, we can now differ between both faulty situations of FCN-*n*. For FCN-*n* with a faulty lieutenant node L_g, both of the corresponding row vector majority and column vector majority (*g*-th row and *g*-th column) will be different from the achieved Byzantine Agreement in node majority. For FCN-*n* with a faulty commander node *C*, the corresponding row vector majority (*g*-th row) will be different from the Byzantine Agreement, however the corresponding column vector majority (*g*-th column) will be the same with the Byzantine Agreement.

6.4 GENERAL CASE OF FCN-*n*

In the previous sections, the faulty node detection of FCN-4 is described. In this section, the general case of FCN-*n* for faulty node detection is explained. In fact, the faulty node detection for $n > 4$ is a straightforward procedure from the case of FCN-4.

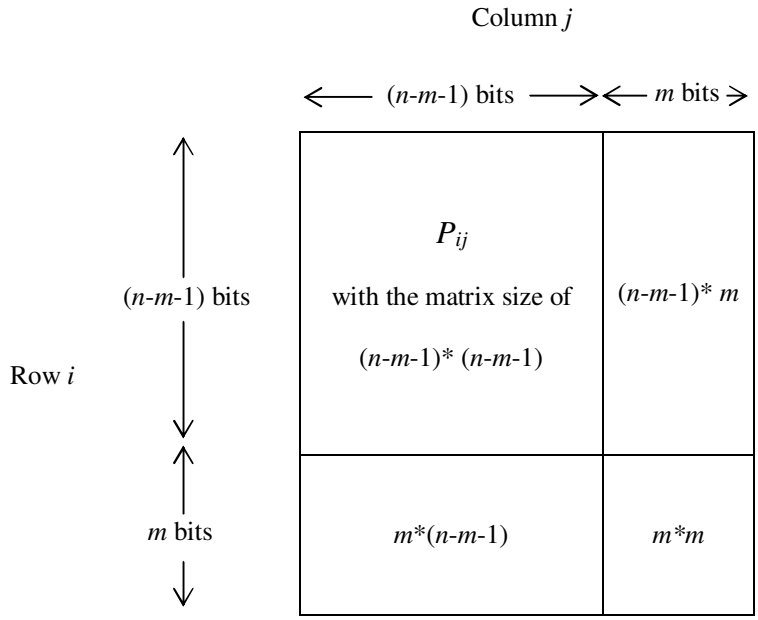


Figure 6.7 MEM of FCN-*n* with its loyal bit and faulty bit composition

For general case of FCN-*n*, the MEM of FCN-4 in Figure 6.1 will be redrawn to become Figure 6.7. The properties of the coordinate P_{ij} of MEM at each node follow the rules in Table 6.1. For FCN-*n* with *m* number of faulty nodes, there are (*n*-*m*-1) rows and (*n*-*m*-1) columns of loyal bit positions. This forms a sub-matrix of (*n*-*m*-1)*(*n*-*m*-1) within the MEM, which is composed of only loyal bit coordinates. For the other three sub-matrices, (*n*-*m*-1)**m*, *m**(*n*-*m*-1) and *m***m*, all of them are composed of faulty bit coordinates.

In other words, we have $(n\text{-}m\text{-}1)^2$ number of loyal bit positions, and $(2mn\text{-}m^2\text{-}2m)$ number of faulty bit positions. For row vector majority, there are (*n*-*m*-1) number of loyal rows. For column vector majority, there are (*n*-*m*-1) number of loyal columns. Let the loyal bit value to be bit "1", then both the cases of faulty commander node and faulty lieutenant node are as illustrated in Figures 6.8 and 6.9 respectively for the worst-case situation of faulty bit composition.

<table>
<tr><td rowspan="3"></td><td colspan="3">MEM</td><td>Local MAJ</td><td>Node MAJ</td></tr>
<tr><td colspan="2">Bit "1"</td><td>Bit "1"</td><td>Bit "1"</td><td rowspan="2">1</td></tr>
<tr><td colspan="2">Bit "0"</td><td>Bit "0"</td><td>Bit "0"</td></tr>
<tr><td>CM</td><td>Bit "1"</td><td>Bit "1"</td><td>Bit "1"</td><td colspan="2"></td></tr>
</table>

- MAJ - majority
- CM - column vector majority

Figure 6.8 Local majority, node majority and column vector majority of the MEM of FCN-*n* with a faulty commander node

<table>
<tr><td rowspan="3"></td><td colspan="3">MEM</td><td>Local MAJ</td><td>Node MAJ</td></tr>
<tr><td colspan="2">Bit “1”</td><td>Bit “0”</td><td>Bit “1”</td><td rowspan="2">1</td></tr>
<tr><td colspan="2">Bit “0”</td><td>Bit “0”</td><td>Bit “0”</td></tr>
<tr><td>CM</td><td>Bit “1”</td><td>Bit “1”</td><td>Bit “0”</td><td colspan="2"></td></tr>
</table>

- MAJ - majority
- CM - column vector majority

Figure 6.9 Local majority, node majority and column vector majority of the MEM of FCN-*n* with a loyal commander node

From the figures above, we can see that the faulty nodes of the general case of FCN-*n* can be detected by using the row vector majority (local majority), column vector majority and node majority. The case of FCN-4, which is discussed in Sections 6.2 and 6.3, is the simplest case for the clarification of the procedures of faulty node detection.

6.5 CONCLUSIONS

The detection of the faulty node within a malicious network will help the loyal nodes to recognize the corrupted message and the malicious sender in the rounds of message exchange. Therefore there is always a need for researcher to discover the algorithm for faulty node identification.

In Section 6.2, the hidden information of message exchange matrix (MEM) is discussed. The relationships of the message coordinate P_{ij} in the MEM, and the identity of message sender are explained in details. In Section 6.3, the faulty node detection of ANN based BAP is realized by applying the majority function to the

MEM in each lieutenant node as many as three times. Both the faulty situations of FCN-*n*, i.e. one with a faulty commander node and one with a faulty lieutenant node, can be differentiated from the comparisons of three majority values: row vector majority, node majority, and column vector majority.

In a nutshell, we can conclude that the faulty node can be detected by examining the row vector majority (or local majority), node majority and column vector majority of the MEM in a particular lieutenant node. This has given the ANN based BAP an ability of allowing the loyal nodes to recognize the identity of the faulty node by running some extra computations.

CHAPTER 7: ANN BASED BAP WITH 3 PARTITIONS

7.1 INTRODUCTION

Artificial neural network (ANN) is introduced to solve the Byzantine Generals Problem (BGP) in Chapter 3. The proposed new approach to BGP is presented in details in Chapter 4 under the title of artificial neural network based Byzantine Agreement Protocol (ANN based BAP). The ANN based BAP has shown better performance over the traditional BAP (Leslie, Shostak & Pease, 1982) when the number of node in a network is equal to or more than ten, $n \geq 10$. The better performance of ANN based BAP is then explained in Chapter 5 via the mathematical analysis on the message exchange matrix (MEM).

In order to improve the performance of the ANN based BAP, another new approach is proposed in this chapter. This new approach is in fact an approach modified from the ANN based BAP. As we can notice in either traditional BAP or ANN based BAP, the n-node network is handled as a single entity without any partition dissecting the n-node network into a few groups of nodes. To be more specific, the ANN based BAP proposed in the previous chapters is known as ANN based BAP with no partition.

In this chapter, the ANN based BAP is dissected into three partitions. This modified ANN based BAP has an n-node network being partitioned into three groups of nodes, and hence it is called ANN based BAP with 3 partitions. The ANN based BAP with 3 partitions shows faster speed in Byzantine Agreement achievement than the ANN based BAP with no partition. This better performance is gained upon the compensation of maximum number of faulty nodes to be tolerated in a malicious network. These few points encircling ANN based BAP with 3 partitions will be discussed in details in the coming sections. Besides, there will be a discussion on ANN based BAP with k partitions and the reason why $k = 3$ is an optimized number of partition. In addition, the faulty node detection for ANN based BAP with 3 partitions is presented as well (Lee & Ewe, 2003a).

7.2 PARTITIONING OF A NETWORK INTO THREE GROUPS OF NODES

For traditional BAP and ANN based BAP with no partition, the smallest network is a 4-node fully connected network (FCN-4). On the other hand, for ANN based BAP with 3 partitions, the smallest network is a 10-node network. In the ANN based BAP with 3 partitions, an *n*-node network is cut into 3 groups of nodes. The difference of the number of nodes between each group is at most one. To understand the protocol better, the denotations below are used throughout the whole chapter.

n = number of nodes in a malicious network

m = number of faulty nodes (worst case of fault situation)

Msg_{3P} = total number of exchanged messages for the case of 3 partitions

$mod(n,3)$ = remainder of (n/3)

ceil(x) = ceiling of x which rounds the elements of x to the nearest integers towards infinity

floor(x) = floor of x which rounds the element of x to the nearest integers towards minus infinity

For an *n*-node network, the number of nodes in each group of the ANN based BAP with 3 partitions depends on the value of $mod(n,3)$. In the process of group partitioning, there are three situations, $mod(n,3)$ = 0, 1 and 2. In the protocol, if all of the three groups are denoted as Group 1, Group 2 and Group 3, the source node or commander node is always located within Group 1. Therefore, Group 1 is always having one node extra or equivalent number of nodes with the other two groups. Table 7.1 shows the relationships of $mod(n,3)$ and number of nodes within a group.

For ANN based BAP with 3 partitions, the worst case of fault situation occurs when only two groups are able to achieve the Byzantine Agreement. The worst situation happens when the faulty nodes concentrate in the same group until the loyal nodes within that group fail to gain the Byzantine Agreement.

Table 7.1 Number of nodes within the groups of the partitioned *n*-node network

mod(n,3)	Group 1	Group 2	Group 3
1	ceil(*n*/3)	floor(*n*/3)	floor(*n*/3)
2	ceil(*n*/3)	ceil(*n*/3)	floor(*n*/3)
0	ceil(*n*/3)	ceil(*n*/3)	ceil(*n*/3)

As an example, the simplest network of ANN based BAP with 3 partitions is illustrated in Figure 7.1, which is a network of 10 nodes. Be aware that within each group there exists a trusted party which is an independent entity computing the group majority or the common agreement among a group. Besides, the source node in Group 1 will become an image of source node for Group 2 and Group 3 during the first round of message exchange of the protocol.

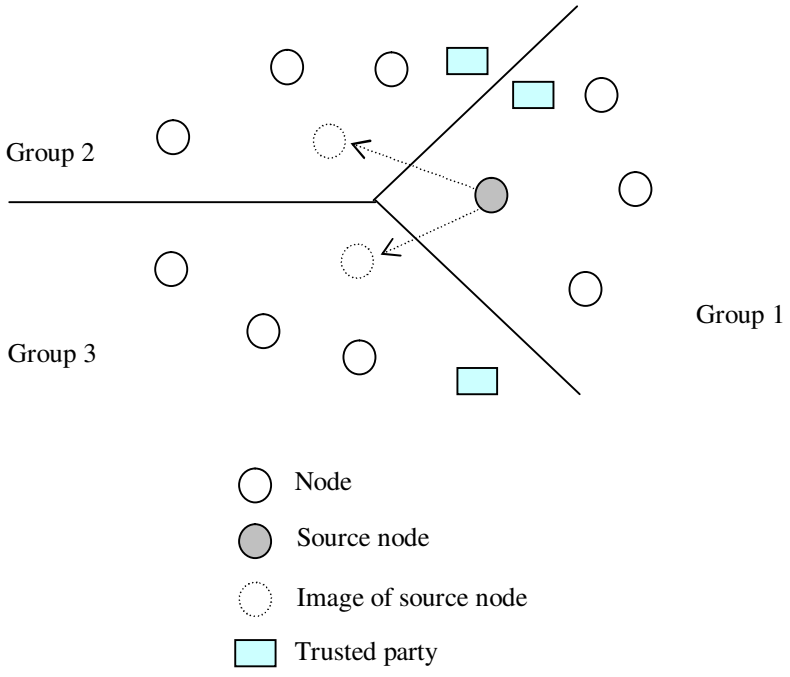


Figure 7.1 A 10-node network cut into 3 groups for ANN based BAP with 3 partitions

7.3 ANN BASED BAP WITH 3 PARTITIONS

The protocol of ANN based BAP with 3 partitions has four rounds of message exchange before the achievement of Byzantine Agreement. Although the ANN based BAP with 3 partitions apparently has one extra round than the ANN based BAP with no partition, it has less number of exchanged messages. This allows the consensus among the loyal nodes to be achieved at faster speed.

To ease the computation, the source node of the message is always put into Group 1. In the first round, the source node emerges as an image source node in the other two groups. The source node indicated by the grey node in Group 1 together with the image nodes in Group 2 and Group 3 act as the message originators. Within each group, the ANN based BAP operates to get the consensus at the group level.

In the second round, every member of a group sends the node decision from the first round to a trusted party in the group. The trusted party will make a *group consensus* by using the majority function. In the third round, the group consensus within the three trusted parties is exchanged among themselves. Based on the exchanged group consensus, a *network consensus* or *global consensus* is made by using the majority function again. In the last round, the three trusted parties will send the achieved global consensus to all the nodes in the network.

For instance, take $n = 10$ as in Figure 7.1. In the first round, there are a total of $3x3^3$ messages being exchanged. In this round, the ANN based BAP with no partition is implemented within each group. There are three batches of intra-group message exchanges. Every batch consists of 3^3 messages within a group.

In the second round, every node within each group sends a message to its corresponding trusted party. Hence, a total of 4, 3, 3 messages are sent from each group to the trusted party 1, 2 and 3 respectively. Then the group consensus is computed at each trusted party. Upon the achievement of the group consensus, it is being exchanged among the trusted parties as in the third round. Six messages are involved here. In the last round, the global consensus is delivered from the trusted parties to all of the nodes. This round involves 10 messages here.

Hence, for $n = 10$, there is a total of 107 messages in the ANN based BAP with 3 partitions as compared to 729 messages for ANN based BAP with no

partition. This is an improvement of 85.32%. However, the ANN based BAP with 3 partitions has its drawback. It can only tolerate for two faulty nodes whereas ANN based BAP with no partition has a tolerance of three faulty nodes. For the same n, this is a decrease of 33.33% of tolerance. The decrease of tolerance for number of faulty nodes is the compensation of having a faster rate of consensus accomplishment.

In general, for an n-node network commencing from $n = 10$, we have the maximum number of faulty nodes m as in Equation (7.1). Meanwhile, the number of exchanged messages at each round and the total exchanged messages are as shown below:

$$m = 2 \text{ x floor}[(\text{ceil}(n/3)-1)/3] = 2 * \lfloor (\lceil n/3 \rceil - 1) /3 \rfloor \quad (7.1)$$

Number of exchanged messages:

First round : $3 \text{ x } [\text{ceil}(n/3)-1]^3$ for $mod(n,3) = 1$ (7.2a)

$2 \text{ x } [\text{ceil}(n/3)-1]^3 + [\text{ceil}(n/3)]^3$ for $mod(n,3) = 2$ (7.2b)

$[\text{ceil}(n/3)-1]^3 + 2 \text{ x } [\text{ceil}(n/3)]^3$ for $mod(n,3) = 0$ (7.2c)

Second round : n messages (7.3)

Third round : 6 messages (7.4)

Fourth round : n messages (7.5)

Total messages:

$Msg_{3P} = 2n + 6 +$ (#message at first round depending on $mod(n,3)$) (7.6)

7.4 EFFECTS OF 3-PARTITION APPROACH TO ANN BASED BAP

To know the effects of 3-partition approach to ANN based BAP with no partition, we need to know the complexity of these two cases. For easier reference, the complexity of ANN based BAP with no partition is repeated in Equation (7.7) from Equation (4.50). To analyze the complexity of ANN based BAP with 3

partitions, we need to investigate the value of Msg_{3P} as the number of nodes n in a malicious network increases towards the infinity.

From Equation (7.6), it can be deduced that the Big-O of Msg_{3P} depends mainly on the number of exchanged messages in the first round instead of the other three rounds. Hence from Equations (7.2a-c), the complexity of the ANN based BAP with 3 partitions is derived to be Equation (7.8). The number of faulty nodes and total number of exchanged messages for ANN based BAP with no partition and 3 partitions are compared in Table 7.2.

$$\text{Complexity of ANN based BAP with no partition} = O(n^3) \tag{7.7}$$

$$\text{Complexity of ANN based BAP with 3 partitions} = O(n^3/9) \tag{7.8}$$

Therefore we can conclude from Equations (7.7) and (7.8) and Table 7.2 that the total number of exchanged messages for ANN based BAP with 3 partitions is about one-ninth of the ANN based BAP with no partition as the network grows larger. This means a decrease of 88.89% from the total number of exchanged messages for ANN based BAP with no partition.

Consequently, in the context of Byzantine Agreement achievement, ANN based BAP with 3 partitions has a faster computational speed because of fewer number of exchanged messages. For better illustration, the total number of exchanged messages for both types of ANN based BAP (with no partition and with 3 partitions), are plotted in Figure 7.2.

Take note that the ANN based BAP with no partition starts from the smallest network of $n = 4$, and the ANN based BAP with 3 partitions starts from the smallest network of $n = 10$. For the comparisons of maximum number of tolerated faulty nodes m, it is in Figure 7.3.

Table 7.2 Total number of exchanged messages, *Msg*, for ANN based BAP with no partition and 3 partitions

n	With no partition		With 3 partitions	
	m	Msg_{ANN}	*m*	Msg_{3P}
10	3	729	2	107
11	3	1000	2	146
12	3	1331	2	185
13	4	1728	2	224
14	4	2197	2	287
15	4	2744	2	350
16	5	3375	2	413
17	5	4096	2	506
18	5	4913	2	599
19	6	5832	4	692
20	6	6859	4	821
21	6	8000	4	950
22	7	9261	4	1079
23	7	10648	4	1250
24	7	12167	4	1421
25	8	13824	4	1592
26	8	15625	4	1811
27	8	17576	4	2030
28	9	19683	6	2249
29	9	21952	6	2522
30	9	24389	6	2795

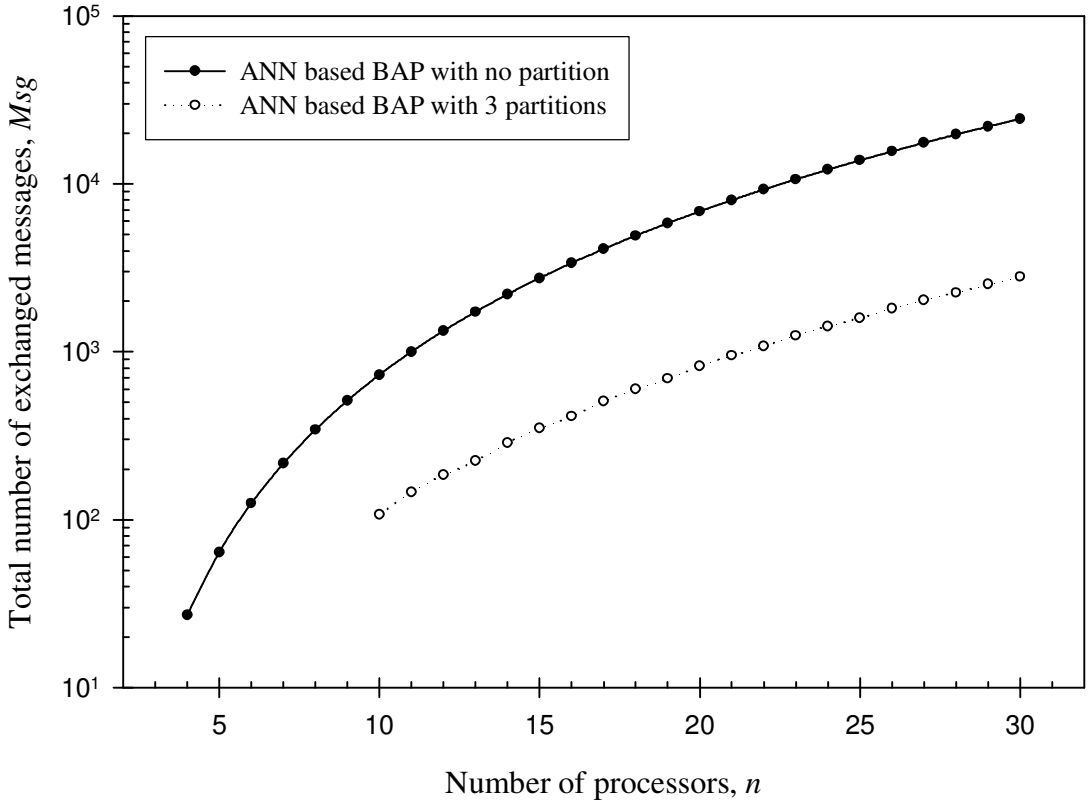


Figure 7.2 Total number of exchanged messages for ANN based BAP with no partition and with 3 partitions

We can discover the effects of increasing network size for ANN based BAP with 3 partitions over the one with no partition. The lesser requirement of exchanged messages allows faster consensus achievement. However there is one disadvantage of ANN based BAP with 3 partitions as from Table 7.2 and Figure 7.3, it can tolerate fewer faulty nodes as compared to ANN based BAP with no partition.

The cycle of m of ANN based BAP with 3 partitions is 9 steps starting from n = 10. It means it needs an increment of 9 units of n for an increment of 2 units of m. On the other hand, the cycle of m of ANN based BAP with no partition is 3 steps starting from n = 4. It takes an increment of 3 units of n for an increment of 1 unit of m. The relationships of cycle of m and increment of m for every cycle can be observed from Equations (2.3) and (7.1). For sufficient large value of n, we can

approximate Equations (2.3) and (7.1) into Equations (7.9) and (7.10) respectively as shown below:

With no partition: $m_{ANN} \approx n/3$ (7.9)

With 3 partitions: $m_{3P} \approx 2n/9$ (7.10)

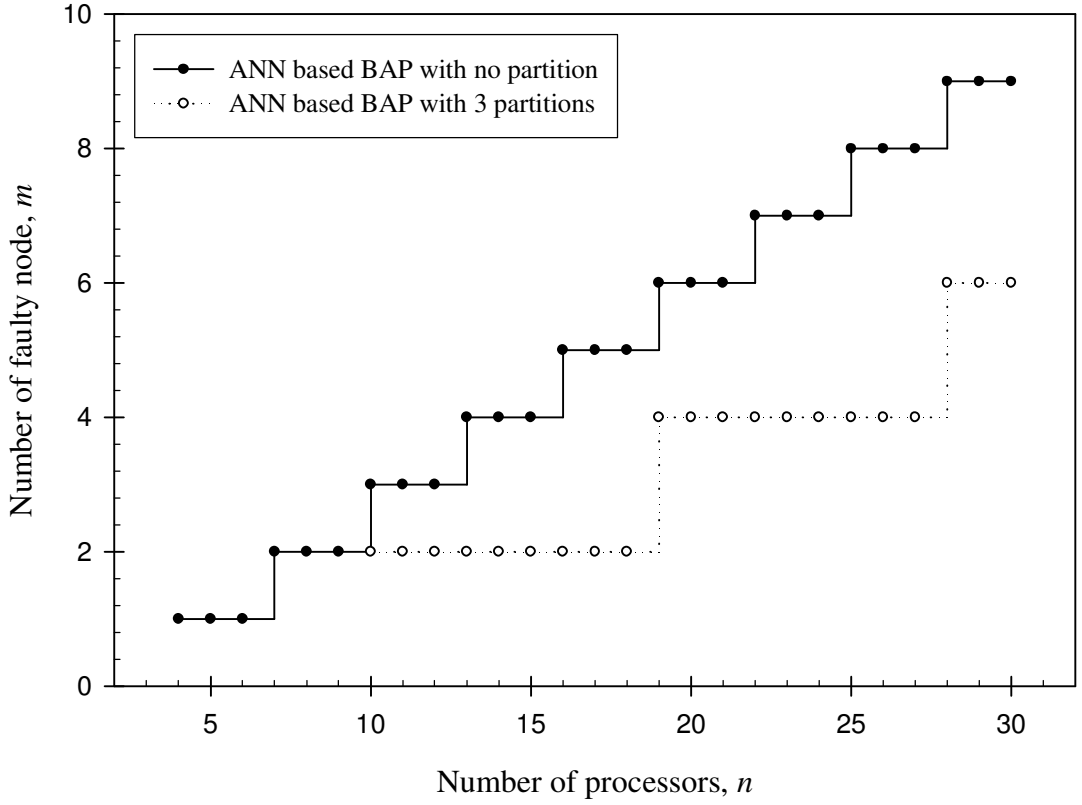


Figure 7.3 Maximum number of tolerated faulty node m for ANN based BAP with no partition and with 3 partitions

The denominators in Equations (7.9) and (7.10) tell the cycle of m, which is the required increment of n before any increment of m. The coefficients of the numerators indicate the increment of m for every cycle of m. As the network becomes big enough, the m ratio of ANN based BAP with 3 partitions to with no partition is 2/3. This means the number of faulty nodes that can be tolerated

decreases for 33.33%. In addition to this disadvantage, another drawback of ANN based BAP with 3 partitions is that it requires a trusted party in every group.

7.5 ANN BASED BAP WITH *k* PARTITIONS

In this section, we will show that the ANN based BAP with 3 partitions is the optimal partition among the *k* partitions. For ANN based BAP with *k* partitions, every partition is by itself a standalone ANN based BAP. Therefore every single partition shall follow the requirement of Equation (2.3) where the minimum number of nodes is $(3m_g+1)$. Assume that m_g is the number of faulty nodes within a group.

For the case of 2 partitions, theoretically the minimum number of nodes is 7 with three nodes and four nodes in each of the two groups. Now consider the worst case of ANN based BAP with 2 partitions where $m = 1$ and the message originator is the faulty node. As in the case of 3 partitions, there are four rounds of communication in the case of 2 partitions. The first and second rounds work well but not the third round. As the protocol comes to the third round, there is no majority when the faulty node is the source node. Each group consensus or group majority may be opposite bit to one another. In other words, the ANN based BAP with 2 partitions fails to solve the BGP by achieving any Byzantine Agreement.

For the case of 4 partitions, the minimum number of nodes is 13 with 3, 3, 3, and 4 nodes in each of the four groups. The worst situation is $m = 2$. It occurs when the message originator is faulty, and one of the group consensus is malicious. The number of faulty nodes *m*, and the number of exchanged messages Msg_{4P} are as shown below:

$$m = 2 \text{ x floor}[(\text{ceil}(n/4)-1)/3] = 2 * \lfloor (\lceil n/4 \rceil - 1) /3 \rfloor \qquad (7.11)$$

Number of exchanged messages:

$$\text{First round : } 4\text{x } [\text{ceil}(n/4)-1]^3 \quad \text{for } mod(n,4) = 1 \qquad (7.12a)$$

$$3\text{x } [\text{ceil}(n/4)-1]^3 + [\text{ceil}(n/4)]^3 \quad \text{for } mod(n,4) = 2 \qquad (7.12b)$$

$$2\text{x } [\text{ceil}(n/4)-1]^3 + 2\text{x } [\text{ceil}(n/4)]^3 \quad \text{for } mod(n,4) = 3 \qquad (7.12c)$$

$$[\text{ceil}(n/4)-1]^3 + 3\text{x}\ [\text{ceil}(n/4)]^3 \quad \text{for } mod(n,4) = 0 \qquad (7.12\text{d})$$

Second round : n messages (7.13)

Third round : 12 messages (7.14)

Fourth round : n messages (7.15)

Total messages:

$$Msg_{4P} = 2n + 12 + (\#\text{message at first round depending on } mod(n,4)) \qquad (7.16)$$

Complexity of ANN based BAP with 4 partitions = $O(n^3/16)$ (7.17)

Cycle of m for ANN based BAP with 4 partitions = 12 (7.18)

Comparing Equations (7.7), (7.8) and (7.17), we expect the ANN based BAP with 4 partitions has a 91.67% decrease of total number of exchanged messages over the ANN based BAP with no partition. From Section 7.4, it is a decrease of 88.89% of total number of exchanged messages for ANN based BAP with 3 partitions over the case of with no partition.

On the other hand, the cycle of m for the case of ANN based BAP with 3 partitions is 9, and it is 12 for the case of ANN based BAP with 4 partitions. It means the number of faulty nodes to be tolerated increases by 2 units for an increment of 9 units and 12 units of n respectively.

Comparing Equations (2.3), (7.1) and (7.11), the case of ANN based BAP with 3 partitions has a drop of 33.33% of maximum faulty nodes to be tolerated over the case of ANN based BAP with no partition. However, the case of ANN based BAP with 4 partitions has a drop of 50% over the case of ANN based BAP with no partition.

Table 7.3 Total number of exchanged messages, *Msg*, for ANN based BAP with 3 partitions and 4 partitions

n	With 3 partitions		With 4 partitions	
	m	Msg_{3P}	*m*	Msg_{4P}
10	2	107	-	-
11	2	146	-	-
12	2	185	-	-
13	2	224	2	146
14	2	287	2	185
15	2	350	2	224
16	2	413	2	263
17	2	506	2	302
18	2	599	2	365
19	4	692	2	428
20	4	821	2	491
21	4	950	2	554
22	4	1079	2	647
23	4	1250	2	740
24	4	1421	2	833
25	4	1592	4	926
26	4	1811	4	1055
27	4	2030	4	1184
28	6	2249	4	1313
29	6	2522	4	1442
30	6	2795	4	1613
31	6	3068	4	1784
32	6	3401	4	1955
33	6	3734	4	2126
34	6	4067	4	2345
35	6	4466	4	2564
36	6	4865	4	2783

Therefore, after the consideration on the factors of number of exchanged messages and cycle of m, it is no doubt that the case of 3 partitions is better than the case of 4 partitions. Table 7.3 shows the number of exchanged messages and number of faulty nodes for ANN based BAP with 3 partitions and 4 partitions. Figure 7.4 shows the number of exchanged messages for ANN based BAP with 3 partitions and 4 partitions. Note that the case of 3 partitions starts from n = 10, and the case of 4 partitions starts from n = 13. Meanwhile Figure 7.5 shows the number of faulty nodes to be tolerated in both of these cases.

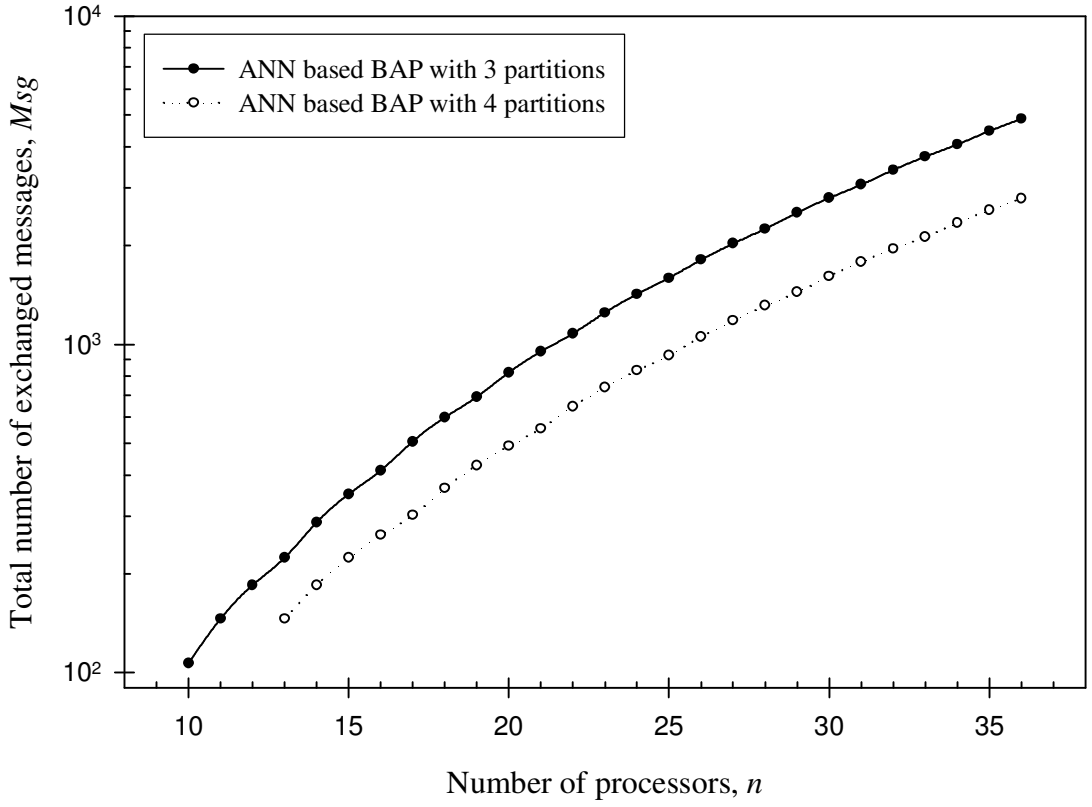


Figure 7.4 Number of exchanged messages for ANN based BAP with 3 partitions and 4 partitions

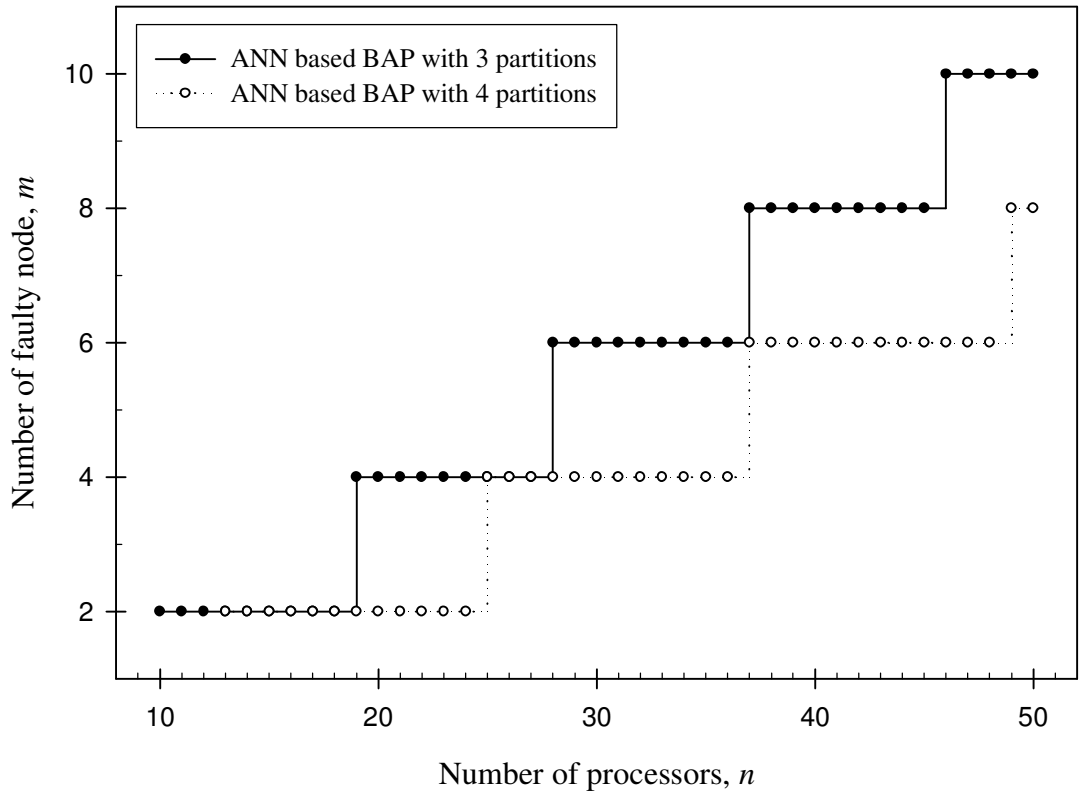


Figure 7.5 Maximum number of tolerated faulty node *m* for ANN based BAP with 3 partitions and 4 partitions

For higher orders of *k*, the number of faulty nodes *m* is expressed in Equation (7.19). The number of exchanged messages for ANN based BAP with *k* partitions, Msg_{kP}, is evaluated in Equations (7.20-7.24). The complexity of ANN based BAP with *k* partitions is expressed in Equation (7.25). Meanwhile Equation (7.26) shows the cycle of *m* for ANN based BAP with *k* partitions.

$$m = 2 * \lfloor (\lceil n/k \rceil - 1) /3 \rfloor = 2 * \lceil n/3k - 1 \rceil \qquad (7.19)$$

Number of exchanged messages:

First round : k x [ceil(n/k)-1]^3 for $mod(n,k) = 1$ (7.20a)

$(k-1)$[ceil(n/k)-1]^3 + [ceil(n/k)]^3 for $mod(n,k) = 2$ (7.20b)

$$(k-2)[\text{ceil}(n/k)-1]^3 + 2\text{x}\,[\text{ceil}(n/k)]^3 \quad \text{for } mod(n,k) = 3 \tag{7.20c}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$(k-r+1)[\text{ceil}(n/k)-1]^3 + (r-1)\text{x}\,[\text{ceil}(n/k)]^3 \quad \text{for } mod(n,k) = r \tag{7.20d}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$3\text{x}\,[\text{ceil}(n/k)-1]^3 + (k-3)[\text{ceil}(n/k)]^3 \quad \text{for } mod(n,k) = k-2 \tag{7.20e}$$

$$2\text{x}\,[\text{ceil}(n/k)-1]^3 + (k-2)[\text{ceil}(n/k)]^3 \quad \text{for } mod(n,k) = k-1 \tag{7.20f}$$

$$[\text{ceil}(n/k)-1]^3 + (k-1)[\text{ceil}(n/k)]^3 \text{ for } mod(n,k) = 0 \tag{7.20g}$$

where $r = 1, 2, 3, \ldots, k-1$

Second round : n messages (7.21)

Third round : $k*(k-1)$ messages (7.22)

Fourth round : n messages (7.23)

Total messages:

$$Msg_{kP} = 2n + k*(k-1) + (\#\text{message at first round depending on } mod(n,k)) \tag{7.24}$$

Complexity of ANN based BAP with k partitions = $O(n^3/k^2)$ (7.25)

Cycle of m for ANN based BAP with k partitions = $3k$ (7.26)

Table 7.4 lists the range of number of nodes n for the ANN based BAP with k partitions, where k = 1, 3, 4, 5, 6, 7, 8, 9 and 10, against number of faulty nodes m, where m = 2, 4, 6 and 8. Note that ANN based BAP with no partition is in fact the case of k = 1. For k = 2, there is no possible solution and hence it is not included. On the other hand, Table 7.5 shows the range of number of exchanged messages *Msg* for the ANN based BAP with k partitions, where k = 1, 3, 4, 5, 6, 7, 8, 9 and 10, against number of faulty node m, where m = 2, 4, 6 and 8. Table 7.6 shows the ratio of *Msg*/m for n = 4, 5, 6, …, 50 against k = 1, 3, 4, 5, 6, 7, 8, 9 and 10.

Table 7.4 Range of number of nodes n for the ANN based BAP with k partitions against number of faulty nodes m

k partitions	Number of faulty nodes, m			
	$m = 2$	$m = 4$	$m = 6$	$m = 8$
1	n = 7-9	13-15	19-21	25-27
3	10-18	19-27	28-36	37-45
4	13-24	25-36	37-48	49-60
5	16-30	31-45	46-60	61-75
6	19-36	37-54	55-72	73-90
7	22-42	43-63	64-84	85-105
8	25-48	49-72	73-96	97-120
9	28-54	55-81	82-108	109-135
10	31-60	61-90	91-120	121-150

Table 7.5 Range of number of exchanged messages Msg for the ANN based BAP with k partitions against number of faulty nodes m

k partitions	Number of faulty nodes, m			
	$m = 2$	$m = 4$	$m = 6$	$m = 8$
1	Msg = 216-512	1728-2744	5832-8000	13824-17576
3	107-599	692-2030	2249-4865	5264-9590
4	146-833	926-2783	3002-6623	7002-13001
5	187-1069	1162-3538	3757-8383	8782-16414
6	230-1307	1400-4295	4514-10145	10544-19829
7	275-1547	1640-5054	5273-11909	12308-23246
8	322-1789	1882-5815	6034-13675	14074-26665
9	371-2033	2126-6578	6797-15443	15842-30086
10	422-2279	2372-7343	7562-17213	17612-33509

Table 7.6 Ratio of *Msg*/*m* for n = 4, 5, 6, …, 50 against *k* = 1, 3, 4, 5, 6, 7, 8, 9 and 10

n	ANN based BAP with *k* partitions (*Msg*/*m*)								
	k = 1	3	4	5	6	7	8	9	10
4	27	-	-	-	-	-	-	-	-
5	64	-	-	-	-	-	-	-	-
6	125	-	-	-	-	-	-	-	-
7	108	-	-	-	-	-	-	-	-
8	171.5	-	-	-	-	-	-	-	-
9	256	-	-	-	-	-	-	-	-
10	243	53.5	-	-	-	-	-	-	-
11	333.3	73	-	-	-	-	-	-	-
12	443.7	92.5	-	-	-	-	-	-	-
13	432	112	73	-	-	-	-	-	-
14	549.3	143.5	92.5	-	-	-	-	-	-
15	686	175	112	-	-	-	-	-	-
16	675	206.5	131.5	93.5	-	-	-	-	-
17	819.2	253	151	113	-	-	-	-	-
18	982.6	299.5	182.5	132.5	-	-	-	-	-
19	972	173	214	152	115	-	-	-	-
20	1143	205.3	245.5	171.5	134.5	-	-	-	-
21	1333	237.5	277	191	154	-	-	-	-
22	1323	269.8	323.5	222.5	173.5	137.5	-	-	-
23	1521	312.5	370	254	193	157	-	-	-
24	1738	355.3	416.5	285.5	212.5	176.5	-	-	-
25	1728	398	231.5	317	232	196	161	-	-
26	1953	452.8	263.8	348.5	263.5	215.5	180.5	-	-
27	2197	507.5	296	395	295	235	200	-	-
28	2187	374.8	328.3	441.5	326.5	254.5	219.5	185.5	-
29	2439	420.3	360.5	488	358	274	239	205	-
30	2710	465.8	403.3	534.5	389.5	305.5	258.5	224.5	-
31	2700	511.3	446	290.5	421	337	278	244	211

32	2979	566.8	488.8	322.8	467.5	368.5	297.5	263.5	230.5
33	3277	622.3	531.5	355	514	400	317	283	250
34	3267	677.8	586.3	387.3	560.5	431.5	348.5	302.5	269.5
35	3573	744.3	641	419.5	607	463	380	322	289
36	3898	810.8	695.8	451.8	653.5	494.5	411.5	341.5	308.5
37	3888	658	500.3	494.5	350	541	443	361	328
38	4221	716.9	545.8	537.3	382.3	587.5	474.5	392.5	347.5
39	4573	775.8	591.3	580	414.5	634	506	424	367
40	4563	834.6	636.8	622.8	446.8	680.5	537.5	455.5	386.5
41	4923	903.3	682.3	665.5	479	727	569	487	406
42	5302	971.9	737.8	720.3	511.3	773.5	615.5	518.5	437.5
43	5292	1041	793	775	543.5	410	662	550	469
44	5679	1120	848.8	829.8	586.3	442.3	708.5	581.5	500.5
45	6085	1199	904.3	884.5	629	474.5	755	613	532
46	6075	1022	970.8	626.2	671.8	506.8	801.5	644.5	563.5
47	6489	1095	1037	671.7	714.5	539	848	691	595
48	6922	1167	1104	717	757.3	571.3	894.5	737.5	626.5
49	6912	1239	877.8	762.7	800	603.5	470.5	784	658
50	7353	1321	936.6	808.2	854.8	635.8	502.8	830.5	689.5

From the comparison of Tables 7.2-7.6, for constant n, we can notice that the decreasing number of exchanged messages is at the expense of the decrease of tolerance for number of faulty nodes. This is because the increasing rate of cycle of m is magnitude-wise faster than the decreasing rate of number of exchanged messages.

In order to have an ANN based BAP with fast rate of consensus accomplishment and high tolerance of faulty nodes, one of the cases of k-partition approaches has to be selected as the optimal partition for the ANN based BAP with k partitions. Between the considerations of less number of exchange messages Msg and number of faulty node m, the factor of faulty node tolerance is more important. Moreover, higher values of k are not in favour practically due to the factor of unknown number of faulty nodes in real situation. The weak fault tolerance of k-

partition approaches with large k subjects to a requirement for high density of loyal nodes, though the speed of consensus accomplishment is faster.

Consequently, a compromise is achieved between the fault tolerance and convergence speed by adopting the case of 3-partition approach. With a reasonable compensation for less tolerance of number of faulty nodes, ANN based BAP with 3 partitions has a significant faster rate of consensus accomplishment than the case of no partition. For the cases of $k > 3$, the speed of consensus accomplishment does not increase significantly as compared to the case of $k = 3$ over $k = 1$ (the case of no partition). At the same time, the cases of $k > 3$ have its fault tolerance significantly drop as compared to the case of $k = 3$ over $k = 1$.

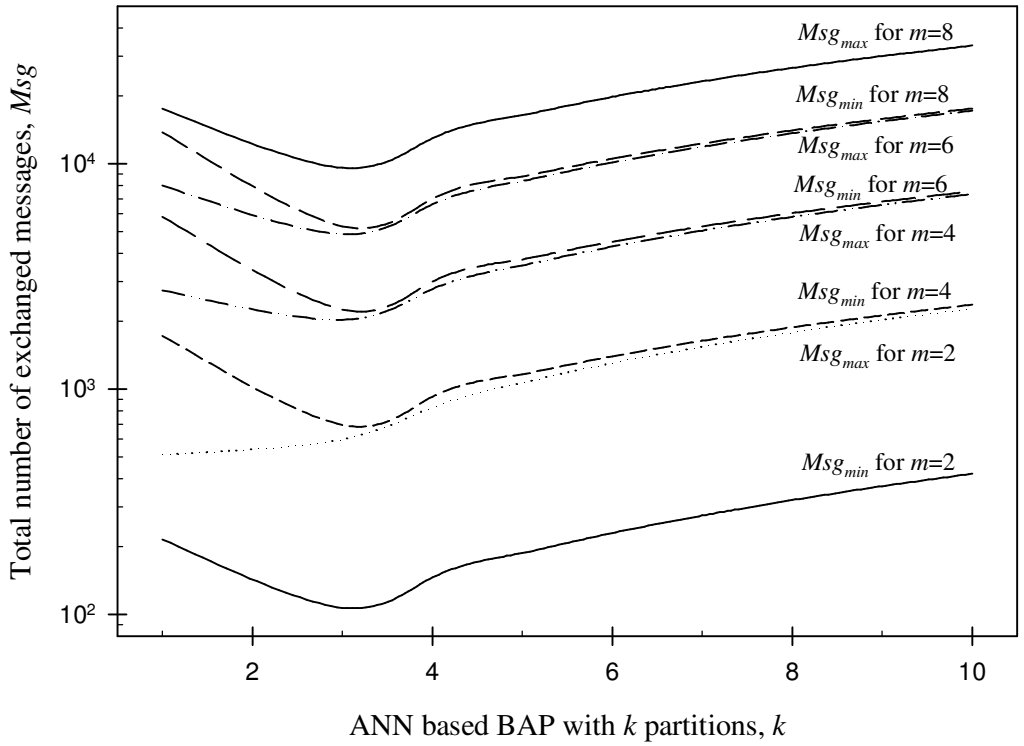


Figure 7.6 Number of exchanged messages Msg against the cases of k-partition approaches for fixed number of faulty nodes m

Figure 7.6 shows the number of exchanged messages Msg against the cases of k-partition approaches for fixed number of faulty nodes m. For every value of m, there exists a range of values for Msg. In the figure, for every m, the minimum and

maximum numbers of exchanged messages (Msg_{min} and Msg_{max}) are plotted to represent the range of Msg for every case of ANN based BAP with k partitions. From the curves of m = 2, 4, 6 and 8, all of them shows that k = 3 is the best choice. Hence, the ANN based BAP with 3 partitions is selected as the optimal protocol for ANN based BAP.

7.6 FAULTY NODE DETECTION FOR ANN BASED BAP WITH 3 PARTITIONS

For ANN based BAP with no partition, computing row vector majority, column vector majority and node majority enables the faulty node to be detected. In this section, the faulty node detection of ANN based BAP with 3 partitions will be discussed.

After the preliminary message exchange process at the first round, a node in a partition knows about the existence of faulty node within its own group. However, the identity of the faulty node is not known. At the end of the fourth round, each node in a partition knows the identity of the faulty node within its own group. This is possible by comparing the global consensus or network consensus sent from the trusted party and the messages received from the other nodes (in the same group) during the first round of message exchange. Nevertheless, each node in a partition is aware of neither the existence nor the identity of the faulty node in the other partitions or groups.

To detect the faulty node in other groups, one extra round of message exchange has to be added in parallel with the second round. Since it is an extra round not required for the achievement of Byzantine Agreement, it is a redundant round merely for the purpose of detecting the faulty node in other groups. We call it as faulty node detection round here.

In the original second round, each node in a partition sends its message to the trusted party within its own group. Concurrently, the faulty node detection round requests every node to deliver its own message to all other nodes in the other partitions. By implementing it, there will be an additional $2n^2/3$ number of exchanged messages as in Equation (7.19). By comparing the global consensus with

the node messages from the other partitions, the faulty node in the other groups can be detected.

$$Msg_{3P}(\text{with faulty node detection}) = Msg_{3P}(\text{no faulty node detection}) + 3(n/3)(2n/3)$$
$$= Msg_{3P}(\text{no faulty node detection}) + 2n^2/3 \quad (7.19)$$

7.7 CONCLUSIONS

The concept to partition the *n*-node network gives a new outlook to the Byzantine Generals Problem (BGP). The ANN based BAP with 3 partitions has shown better performance over the ANN based BAP with no partition. It has faster speed to achieve the common consensus among the loyal nodes. Nevertheless, the decrease of the number of exchanged messages is gained upon the compensation of less faulty nodes to be tolerated.

In Section 7.2, the criteria to partition an *n*-node network are listed together with a given example on a 10-node distributed system. In Section 7.3, the ANN based BAP with 3 partitions is discussed in details. The number of faulty nodes and the number of exchanged messages are included as well. In Section 7.4, the effects of partitioning over the ANN based BAP is analyzed. It shows that the ANN based BAP with 3 partitions has less requirement for the number of exchanged messages to arrive at the common agreement as compared to the ANN based BAP with no partition. It is an improvement of 88.89% when the network is sufficiently big. On the contrary, there is a drawback of ANN based BAP with 3 partitions. Smaller number of faulty nodes can be tolerated as compared to the normal case of with no partition.

In Section 7.5, it is the discussion on the optimal partitioning of ANN based BAP. Upon the consideration of *k* partitions, few cases of partitioning are analyzed. For the case of 2 partitions, it is shown that it is an impossible implementation when the worst situation occurs. For 3 partitions and 4 partitions, both of the critical factors on number of exchanged messages and number of tolerated faulty nodes are compared. It is proven that the case of 3 partitions is more efficient than the case of 4 partitions. Then when higher orders of *k* partitions are pondered, it is found that in term of magnitude, the rate of decrease of number of exchanged messages is slower

than the rate of decrease of tolerated faulty nodes. Consequently, ANN based BAP with 3 partitions is an optimized partition.

In Section 7.6, the faulty node detection of ANN based BAP with 3 partitions is discussed. At the end of the last round of the original protocol, each node knows the identity of the faulty node within the same group or partition, but not the faulty node within the other groups. By adding an extra round in parallel with the second round of the original protocol, every node in a partition knows the identity of the faulty node within the other partitions. This research result is then documented in Lee and Ewe (2003d).

CHAPTER 8: CONCLUSIONS

8.1 SUMMARY AND CONCLUSIONS OF THE RESEARCH PROJECT

The existence of arbitrary faults in a computing system had become an interesting topic of research almost in parallel with the advent of computer technologies in the twentieth century. The common consensus problem becomes more and more critical and important when there are more and more computing devices being linked to form a network of computation. The breakthrough of cryptography in the end of 1970s had cleared the obstacles towards the path of solution. The introduction of the concept of public key cryptosystem (Diffie & Hellman, 1976) and the realization of this concept via the RSA cryptosystem (Rivest, Shamir & Adleman, 1978) made a giant advancing step towards the tolerance of arbitrary faults.

In the early 1980's, lots of researches were encircling this issue. This problem was then commonly known as Byzantine Generals Problem (BGP) by Lamport, Shostak and Pease (1982) together with its solutions for the unsigned message system and signed message system. The solution to BGP is then called Byzantine Agreement Protocol (BAP). Since that significant juncture, many alternative BAPs have been proposed to reduce the high costs of traditional BAP in terms of memory space and computational speed requirement. The development of these BAPs is briefed in the Table 1.1. All of them deal on the different assumptions of network topologies, processor faults and link faults.

Nevertheless, when an extremely high reliability system is required, the full expense of a BGP solution is needed. All of these BAP variants have to be shifted back to the traditional BAP by Lamport et al. (1982). In 2001, Wang and Kao proposed a new approach to BGP by using the simple artificial neural network (ANN). The application of ANN has shown the advantages of overcoming the huge demands for memory space and computing speed. In the context of this research project, some improvements are done over the Wang and Kao's approach (2001) to have faster convergence speed of ANN training and higher rate of consensus accomplishment by using the Nguyen-Widrow initialization and modified BPNN.

The improved protocol of this thesis is called artificial neural network based Byzantine Agreement Protocol (ANN based BAP). Besides, faulty node detection and the innovative concept of ANN based BAP with partitions is proposed. It is found that the ANN based BAP with 3 partitions is the optimal approach.

This research project started with the literature survey and review on the distributed systems, concept of dependability, fault classification and fault tolerance. All of these topics are discussed in Chapter 2. Besides, this chapter also includes the solution to the universal fault class called Byzantine fault. The BGP and the traditional BAP is explained together with some solid examples. Moreover the mathematical analysis is carried out over the total number of message exchange and computational complexity of the traditional BAP.

In Chapter 3, the integration of ANN into the traditional BAP is discussed. In Chapter 4, the ANN based BAP is explained in depth from the aspects of implementation, complexity and advantages. A few experiments have been run to evaluate the applicability of ANN based BAP over a fully connected network (FCN) in the existence of arbitrary faults. The generals' message system of unsigned message or oral message is used. The protocol of ANN based BAP is divided into five stages: initialization stage, message exchange stage, training stage, application stage and compromise stage.

Compared to the Wang and Kao's approach (2001), the proposed ANN based BAP in this thesis consumes three rounds of message exchange instead of (m+1) rounds in the message exchange stage (m denotes the number of faulty nodes). Meanwhile, in the training stage, the gradient descent based back propagation neural network (BPNN) is used as the learning algorithm for the embedded neural network. Modified BPNN is introduced to gain faster speed of convergence as compared to standard BPNN. The analysis of ANN based BAP on the number of exchanged messages and computational complexity shows that its performance is better than traditional BAP when the size of network $n \geq 10$. In addition, the inherent properties of ANN have integrated many advantages into the ANN based BAP. These improvements are less memory space demand, parallel processing at each node, and

the dynamic learning ability of neural networks to the flexible and versatile Byzantine environment.

In Chapter 5, the research results on the mathematical analysis of message exchange matrix (MEM) are presented. MEM is a matrix formed in the message exchange stage. It is used as the training set in the training stage and testing set in the application stage of the ANN based BAP. The analysis shows that the required mean epoch of neural network training decreases as the network size is increasing with fixed number of faulty nodes. On the other hand, the MEM is applied to detect the faulty node. The faulty node detection is made possible by using the row vector majority (or local majority), node majority (or Byzantine Agreement) and column vector majority on the MEM matrix. This topic is discussed in Chapter 6.

So far, the discussed ANN based BAP is a protocol over a network without any partition. In Chapter 7, an innovative approach is taken by partitioning the *n*-node network into three groups of nodes. This modified ANN based BAP is called ANN based BAP with 3 partitions as compared to the ANN based BAP with no partition in Chapters 4-6. By considering the cases of *k* partitions, the case of 3 partitions has shown to be an optimal solution. The analysis is also carried out over the number of message exchange and computational complexity of the ANN based BAP with 3 partitions. For sufficient big network, it is found that the ANN based BAP with 3 partitions has almost 90% performance improvement in computing Byzantine Agreement over the ANN based BAP with no partition with the compensation of fewer faulty nodes to be tolerated.

8.2 SUGGESTIONS FOR FUTURE RESEARCH

The ANN based BAP proposed in this research project applies to a fully connected network (FCN). There is assumption that the link fault is classified as a processor fault as well. Moreover, all the faults are considered as arbitrary faults. The assumptions of FCN, processor faults and arbitrary faults make the ANN based BAP to be a genaralized BAP. For system with lower reliability requirement, making specialized network model can further reduce the computational task. For the network topology, instead of FCN, there are non-FCN and broadcast network. For

processor faults, instead of arbitrary faults, the model can have dormant faults and hybrid faults. For link faults, there are arbitrary faults and hybrid faults. Hence, some variants of the generalized ANN based BAP for more specialized fault models could be the next gateway of research for applying the ANN into the BGP.

From Hassibi, Stork and Wolff (1993), the trained neural network could be pruned to remove the unimportant weights. This may improve the generalization abilities and storage requirements of the ANN based BAP. In addition, Fiesler and Beale (1997) may help to further ameliorate the ANN generalization towards an optimal ANN design in some prospects. For instances, these prospects are neural network topologies, neural network training, network analysis techniques, radial basis function (RBF) neural networks, neuro-fuzzy systems, neuro-evolutionary systems and the neural network hardware implementations.

This thesis is carried out over the BGP of unsigned message system. Hence, further research can be done for BGP of signed message system. Moreover, there are rooms of trying other ANN models for solving the BGP. This is workable in the purpose of gaining a faster speed in Byzantine Agreement achievement. Besides, other optimization approaches, such as fuzzy logic, genetic algorithm, and evolutionary programming, shall be surveyed and investigated.

Lastly, the advance of Internet technologies has led to an *e*-life based human society with abundant use of *e*-applications in communications, *e*-commerce, *e*-education and *e*-governance. The number of involvement is no longer limited to one or two parties. Hence the established cryptographic models of secret key cryptosystem and public key cryptosystem are no longer sufficient as well. This is because the required protocol in an Internet environment involves not only single party or two parties, but it involves a multi-party circumstance.

These phenomena have triggered the demand for multi-party protocol. To secure the multi-party protocol, a new branch of cryptology is developed. It is known as secure multi-party protocol or distributed cryptography. BGP is by nature a unanimous multi-party problem in order to gain common consensus. Hence, the solution of BGP, such as ANN based BAP, could be helpful in the development of distributed cryptography.

REFERENCES

Aguilera, M. K., & Toueg, S. (1999). A simple bivalency proof that *t*-resilient consensus requires *t*+1 rounds. Information Processing Letters 71(3-4), 155-158.

Alstein, D. (1996). Distributed algorithms for hard real-time systems. Doctoral dissertation, Eindhoven University of Technology, Netherlands.

Anderson, T., & Lee, P. A. (1981). Fault tolerance principles and practice. Englewood Cliffs, New Jersey: Prentice Hall.

Anderson, J. A., Silverstein, J. W., Ritz, S. A., & Jones, R. S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psychological Review, 84, 413-451.

Avizienis, A. (1976). Fault-tolerant systems. IEEE Transactions on Computers, C-25(12), 1304-1312.

Babaoglu, O. & Drummond, R. (1985). Streets of Byzantium: Network architectures for fast reliable broadcasts. IEEE Transactions on Software Engineering, 11(6), 546-554.

Barborak, M., Malek, M., & Dahbura, A. (1993). The consensus problem in fault-tolerant computing. ACM Computing Survey, 25(2), 171-220.

Bolt, G. R. (1992). Fault tolerance in artificial neural networks. Doctoral dissertation, York University, Ontario, Canada.

Bose, N. K., & Liang, P. (1996). Neural network fundamentals with graphs, algorithms, and applications. New York, NY: McGraw-Hill.

Buchanan, B. G., & Shortliffe, E. H. (Eds.). (1984). Rule-based expert systems. Reading, Massachusetts: Addison-Wesley.

Castro, M. (2001). Practical Byzantine fault tolerance. Doctoral dissertation, Massachusetts Institute of Technology, Massachusetts.

Cheung, Raymond K. M., Lustig I., & Kornhauser, A. L. (1990). Relative effectiveness of training set patterns for backpropagation. IEEE International Joint Conference on Neural Networks (IJCNN), 1, 673-678.

Christian, F., Aghili, H., Strong, H. R. (1985). Atomic broadcast: From simple message diffusion to Byzantine agreement. Proceedings of the 15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15), 200-206.

Churchland, P. S., & Sejnowski, T. J. (1992). The computational brain. Cambridge, Massachusetts: MIT Press.

Cooke, M. J., & Lebby, G. L. (1998). An optimal design for mulitlayer feedforward networks. Proceedings of the IEEE 30th Southeastern Symposium on System Theory, 507-511.

Coulouris, G., Dollimore, J., & Kindberg, T. (1994). Distributed systems: Concepts and design (2nd ed.). Essex, England: Addison-Wesley.

Coulouris, G., Dollimore, J., & Kindberg, T. (2001). Distributed systems: Concepts and design (3rd ed.). Essex, England: Pearson Education.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. Communications of the ACM, 34(2), 56-78.

Cristian, F., Aghili, H., Strong, R., & Dolev, D. (1985). Atomic broadcast: From simple message diffusion to Byzantine Agreement. Information and Computation, 118, 158-179.

Demuth, H., & Beale, M. (1992-2000). Neural network toolbox user's guide (version 3 & 4). Natick, Massachusetts: The MathWorks.

DeRouin, E., Brown, J., Beck, H., Fausett, L., & Schneider, M. (1991). Neural network training on unequally represented classes. In C. H. Dagli, S. R. T. Kumara, & Y. S. Shin (Eds.), Intelligent Engineering Systems Through Artificial Neural Networks (pp. 135-141). New York: ASME Press.

Diffie, W., & Hellman, M. E. (1976). New directions in cryptography. IEEE Transactions on Information Theory, IT-22(6), 644-654.

Dolev, D. (1982). The Byzantine Generals strike again. Journal of Algorithms, 3(1), 14-30.

Faggin, F. (1991). VLSI implementation of neural networks: Tutorial notes. International Joint Conference on Neural Networks. Seattle, WA.

Farley, B. G., & Clark, W. A. (1954). Simulation of self-organisation systems by digital computer. Institute of Radio Engineers -- Transactions of Professional Group of Information Theory (PGIT-4), 4(3), 76-84.

Fausett, L. (1994). Fundamentals of neural networks: Architectures, algorithms and applications. Englewood Cliffs, New Jersey: Prentice Hall.

Feldman, P. N. (1988). Optimal algorithms for Byzantine Agreement. Doctoral dissertation, Massachussets Institute of Technology, Massachussets.

Feldman, J., & Ballard, D. (1982). Connectionist models and their properties. Cognitive Science, 6(3), 205-254.

Fiesler, E., & Beale, R. (Eds.). (1997). Handbook of neural computation. Joint publication of Bristol: Institute of Physics Publishing & Oxford: Oxford University Press.

Fischer, M. J., & Lynch, N. A. (1982). A lower bound for the time to assure interactive consistency. Information Processing Letters, 14(4), 183-186.

Fu, L. M. (1994). Neural networks in computer intelligence. New York, NY: McGraw-Hill.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. Neural Computation 4, 1-58.

Goldwasser, S. (1997). New directions in cryptography: Twenty some years later. Proceedings of the 38th IEEE Annual Symposium on Foundations of Computer Science, 314-324.

Grossberg, S. (1980). How does a brain build cognitive code? Psychological Review, 87, 1-51.

Gurney, K. (1997). An introduction to neural networks. London: University College London (UCL).

Hagan, M. T., & Demuth, H. B. (1995). Neural network design. Boston, Massachusetts: PWS Publishing Company.

Hassibi, S., Stork, D. G., & Wolff, G. J. (1993). Optimal brain surgeon and general network pruning. Proceedings of the IEEE International Conference on Neural Networks 1993, 1, 293-299.

Haykin, S. (1994). Neural networks: A comprehensive foundation. Don Mills, Ontario: Macmillan College Publishing Company.

Hebb, D. O. (1949). The organisation of behaviour: A neuropsychological theory. New York, NY: John Wiley.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of USA, 72, 2554-2558.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. Neural Networks, 1(4), 295-307.

Jalote, P. (1994). Fault tolerance in distributed systems. Englewood Cliffs, New Jersey: Prentice Hall.

Kandel, A., & Langholz, G. (Eds.). (1992). Hybrid architectures for intelligent systems. Boca Raton, Florida: CRC Press.

Kaufman, C., Perlman, R., & Speciner, M. (1995). Network security: Private communication in a public world. Upper Saddle River, New Jersey: Prentice Hall.

Kohonen, T. (1977). Associative memory: A system-theoretical approach. New York: Springer-Verlag.

Krol, T. (1991). A generalization of fault-tolerance based on masking. Doctoral dissertation, Eindhoven University of Technology, Netherlands.

Lamport, L. (1983). The weak Byzantine Generals Problem. Journal of ACM, 30(3), 228-234.

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3), 382-401.

Laprie, J. C. (1985). Dependable computing and fault-tolerance: Concepts and terminology. Proceedings of the 15th IEEE International Symposium on Fault-Tolerant Computing Systems (FTCS-15), 2-11.

Laprie, J. C. (1992). Dependability: Basic concepts and terminology - In English, French, German, and Japanese. Vienna: Spring-Verlag.

Laprie, J. C. (1995). Dependability - Its attributes, impairments and means. In B. Randell, J. C. Laprie, H. Kopetz, & B. Littlewood (Eds.), Predictably Dependable Computing Systems (pp. 1-28). Berlin: Spring-Verlag.

Laranjeiri, L., Malek, M., & Jenevein, R. (1991). On tolerating faults in naturally redundant algorithms. Proceedings of the 10th IEEE Symposium on Reliable Distributed Systems, 118-127.

Lee, K. W., & Ewe, H. T. (2001). Artificial neural networks based algorithm for Byzantine Generals Problem. MMU International Symposium on Information and Communications Technologies 2001, 1.4(1-4).

Lee, K. W., & Ewe, H. T. (2002a). Artificial neural network based Byzantine agreement protocol. Proceedings of the 6th IASTED International Conference on Artificial Intelligence and Soft Computing, 368-373.

Lee, K. W., & Ewe, H. T. (2002b). Analysis of message exchange matrix of artificial neural network based Byzantine Agreement Protocol. Manuscript submitted for publication, Multimedia University, Malaysia.

Lee, K. W., & Ewe, H. T. (2003a). Artificial neural networks based Byzantine Agreement Protocol with 3 Partitions. Manuscript submitted for publication, Multimedia University, Malaysia.

Lee, K. W., & Ewe, H. T. (2003b). Faulty node detection in the ANN based BAP. Manuscript submitted for publication, Multimedia University, Malaysia.

Lee, K. W., & Ewe, H. T. (2003c). Improving the convergence speed of artificial neural networks based Byzantine Agreement Protocol. Manuscript in progress, Multimedia University, Malaysia.

Lee, K. W., & Ewe, H. T. (2003d). Faulty node detection in the tripartite ANN based BAP. Manuscript in progress, Multimedia University, Malaysia.

Lincoln, P., & Rushby, J. (1993). A formally verified algorithm for interactive consistency under a hybrid fault model. Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23), 402-411.

Lursinsap, C., & Kim, J. H. (1991). Parallel learning for back-propagation network in binary field. IEEE International Symposium on Circuits and Systems, 3, 1477-1480.

MATLAB: The language of technical computing (version 5). (Rev. ed.). (1997). Natick, Massachusetts: The MathWorks.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of ideas imminent in nervous activity. Bulletin of Mathematics and Biophysics, 5, 115-133.

Mead, C. A., & Conway, L. (1980). Introduction to VLSI systems. Reading, Massachusetts: Addison-Wesley.

Meyer, F. J., & Pradhan, D. K. (1991). Consensus with dual failure modes. IEEE Transactions on Parallel and Distributed Systems, 2(2), 214-222.

Minsky, M., & Papert, S. (1969). Perceptrons. Cambridge, Massachusetts: MIT Press.

Mishra, S., & Schlicthing, R. D. (1992). Abstractions for constructing dependable distributed systems (Tech. Rep. TR 92-19). Tucson, Arizona: University of Arizona, Department of Computer Science.

Nguyen, D., & Widrow, B. (1990). Improving the learning speed of two-layer neural networks by choosing initial values of the adaptive weights. Proceedings of the IEEE International Joint Conference on Neural Networks, 3, 21-26.

O'Hare, G. M. P., & Jennings, N. R. (Eds.). (1996). Foundations of Distributed Artificial Intelligence. New York, NY: John Wiley.

Parker, D. B. (1982). Learning logic (Invention Report S81-64, File 1). Stanford, California, United States: Stanford University, Office of Technology Licensing.

Pease, M., Shostak, R., & Lamport, L. (1980). Reaching agreement in the presence of faults. Journal of ACM, 27(2), 228-234.

Perlman, R. (1988). Network layer protocols with Byzantine robustness. Doctoral dissertation, Massachusetts Institute of Technology, Massachusetts.

Postma, A. (1998). Classes of Byzantine fault-tolerant algorithms for dependable distributed systems. Doctoral dissertation, Twente University, Netherlands.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM, 21(2), 120-126.

Rooij, A. J. F. V., Jain, L. C., & Johnson, R. P. (1996). Neural network training using genetic algorithms. Farrer Road, Singapore: World Scientific Publishing.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organisation in the brain. Psychological Review, 65, 386-407.

Rumerhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representation by error propagation. In Parallel distributed processing: Explorations in the microstructures of cognition: Vol. 1 (318-362). Cambridge, Massachusetts: MIT Press.

Rumelhart, D. E., McClelland, J. L., & the PDP Research Group (Eds.). (1986). Parallel distributed processing: Explorations in the microstructures of cognition (Vol. 1-2). Cambridge, MA: MIT Press.

Schalkoff, R. (1997). Artificial neural networks. New York, NY: McGraw-Hill.

Schlichting, R., & Schneider, F. B. (1983). Fail-stop processors: An approach to designing fault-tolerant computing systems. ACM Transactions on Computer Systems, 1(3), 222-238.

Schneier, B. (1996). Applied cryptography (2nd ed.). New York, NY: John Wiley & Sons.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. Complex Systems, 1, 145-168.

Shin, K., & Ramanathan, P. (1987). Diagnosis of processors with Byzantine faults in a distributed computing system. Proceedings of the 17th International Symposium on Fault-Tolerant Computing (FTCS-17), 55-60.

Siewiorek, D. P., & Swarz, R. S. (1998). Reliable computer systems: Design and evaluation (3rd ed.). Natick, Massachusetts: A K Peters.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2002). Operating system concepts (6th ed.). New York, NY: John Wiley & Sons.

Simon, B. P., & Eswaran, C. (1997). An ECG classifier designed using modified decision based neural networks. ACM Transactions on Computers and Biomedical Research, 30(4), 257-272.

Siu, H. S., Chin, Y. H., & Yang, W. P. (1998). Byzantine agreement in the presence of mixed faults on processors and links. IEEE Transactions on Parallel and Distributed Systems, 9(4), 335-345.

Smolensky, P. (1988). On the proper treatment of connectionism. Behavioural and Brain Sciences, 11, 145-168.

Snider, L. A., & Yuen, Y. S. (1997). The artificial neural networks based relay algorithm for distribution system high impedance fault detection. Proceedings of the 4th IEEE International Conference on Advances in Power System Control, Operation and Management (APSCOM-97), 100-106.

Stallings, W. (1999). Cryptography and network security: Principles and practice (2nd ed.). Upper Saddle River, New Jersey: Prentice Hall.

Swingler, K. (1996). Applying neural networks: A practical guide. London: Academic Press Limited.

Thambidurai, P., & Park, Y. K. (1988). Interactive consistency with multiple failure modes. Proceedings of the 7th IEEE Symposium on Reliable Distributed Systems, 93-100.

Trivedi, K. S. (1982). Probability and statistics with reliability, queuing, and computer science applications. Englewood Cliffs, New Jersey: Prentice Hall.

Wang, S. C., & Kao, S. H. (2001). A new approach for Byzantine agreement. Proceedings of the 15th IEEE International Conference on Information Networking, 518-524.

Wang, S. C., & Yan K. Q. (2000). Reaching fault diagnosis agreement on dual link failure mode. Proceedings of the 7th IEEE International Conference on Parallel and Distributed Systems, 291-298.

Werbos, P. J. (1974). Beyond regression: New tools for prediction and analysis in the behavioural sciences. Doctoral dissertation, Harvard University, Cambridge, Massachusetts, United States.

Widrow, B., & Hoff, M. E., Jr. (1960). Adaptive switching circuits. IRE WESCON Convention Record, 96-104.

Yan, K. Q., & Chin, Y. H. (1988). An optimal solution for consensus problem in an unreliable communication system. Proceedings of the International Conference on Parallel Processing, 388-391.

Yan, K. Q., Chin, C. H., & Wang, S. C. (1992). Optimal agreement protocol in malicious faulty processors and faulty links. IEEE Transactions on Knowledge and Data Engineering, 4(3), 266-280.

Yan, K. Q., Wang, S. C., & Chin, Y. H. (1999). Consensus under unreliable transmission. Information Processing Letters 69, 243-248.